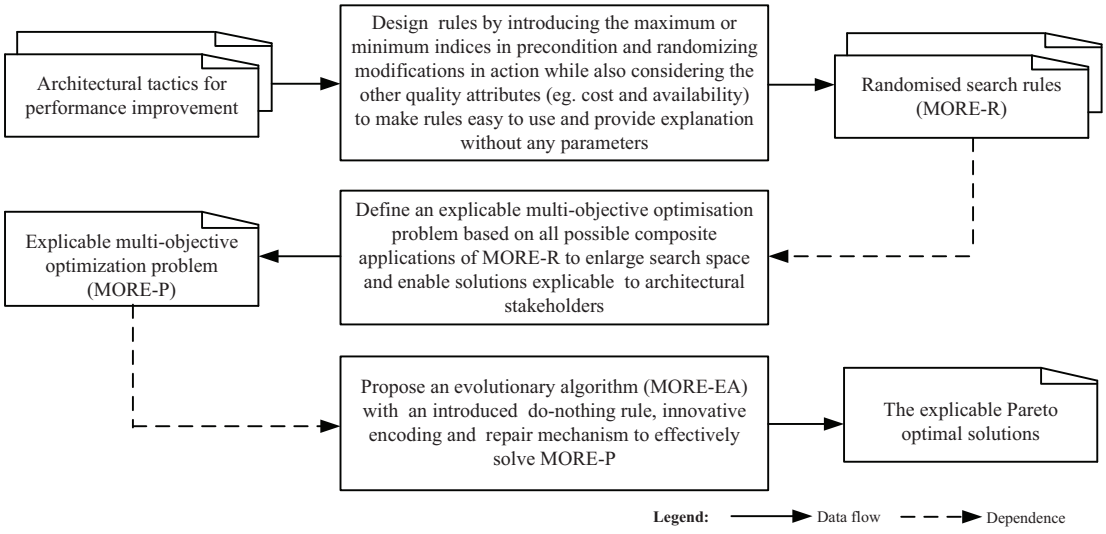
[COR: alpha] [COR: alpha]

Graphical Abstract

Multi-Objective Software Performance Optimisation at the Architecture Level Using Randomised Search Rules

Youcong Ni,Xin Du,Peng Ye,Leandro L. Minku,Xin Yao,Mark Harman,Ruliang Xiao



The overall framework of MORE proposed in this paper

Highlights

Multi-Objective Software Performance Optimisation at the Architecture Level Using Randomised Search Rules

Youcong Ni,Xin Du,Peng Ye,Leandro L. Minku,Xin Yao,Mark Harman,Ruliang Xiao

- Rule methods exclude good solutions owing to preset parameters existed in rule.
- Metaheuristic-based methods lack explicability due to ignoring practical knowledge.
- Integrating rule with metaheuristic methods can obtain more explicable and higher quality solutions.

Multi-Objective Software Performance Optimisation at the Architecture Level Using Randomised Search Rules

Youcong Ni[a,c], Xin Du[a,c,**], Peng Ye[b,*], Leandro L. Minku[d], Xin Yao[e], Mark Harman[f] and Ruliang Xiao[a]

[a]College of Mathematics and Informatics, Fujian Normal University, Fuzhou 350117, China
[b]College of Mathematics and Computer, Wuhan Textile University, Wuhan 430200, China
[c]Fujian Provincial Engineering Technology Research Center for Public Service Big Data Mining and Application, Fujian Normal University, Fuzhou 350007, Fujian, China
[d]Department of Informatics, University of Leicester, Leicester, LE1 7RH, United Kingdom
[e]Department of computer science and engineering, Southern University of Science and Technology, Shenzhen, 518055, China
[f]Department of Computer Science, University College London, London, WC1E 6BT, United Kingdom

ARTICLE INFO

Keywords:
Software Architecture
Performance Optimisation
Multi-objective Evolutionary Optimisation
Randomised Search Rule

Abstract

Architecture-based software performance optimisation can help to find potential performance problems and mitigate their negative effects at an early stage. To automate this optimisation process, rule-based and metaheuristic-based performance optimisation methods have been proposed. However, existing rule-based methods explore a limited search space, potentially excluding optimal or near-optimal solutions. Most of current metaheuristic-based methods ignore existing practical knowledge of performance improvement, and lead to solutions that are not easily explicable to humans. To address these problems, we propose a novel approach for performance optimisation at the software architecture level named Multiobjective performance Optimisation based on Randomised search rulEs (MORE). First, we design randomised search rules (MORE-R) to provide explanation without parameters while benefiting from existing practical knowledge of performance improvement. Second, based on all possible composite applications of MORE-R, an explicable multi-objective optimisation problem (MORE-P) is defined to enlarge search space and enable solutions explicable to architectural stakeholder. Third, a multi-objective evolutionary algorithm (MORE-EA) with an introduced do-nothing rule, innovative encoding and repair mechanism is designed to effectively solve MORE-P. The experiments show that MORE is able to achieve more explicable and higher quality solutions than two state-of-the-art techniques. They also demonstrate the benefits of integrating search-based software engineering approaches with practical knowledge.

E-mail addresses: youcongni@foxmail.com (Youcong Ni), xindu79@126.com (Xin Du), whuyp@126.com (Peng Ye)
*Corresponding author
**Principal corresponding author
ORCID(s):

List of Acronyms

DoF	Degree of freedom
LQN	Layered Queueing Network
MORE	Multi-objective performance Optimisation based on Randomised search rulEs
MORE-R	Randomized search rules
MORE-P	MORE optimisation Problem
MORE-EA	Evolutionary Algorithm to solve MORE-P
PB	Performance Booster
PCM	Palladio Component Model
SA	Software Architecture

1. Introduction

Performance is not only an important quality attribute of software systems (Chen et al., 2018), but also a vital factor to determine success or failure of a system. Cost and risk can be prominently reduced when performance problems (e.g., high resource utilisation, long response time and low throughput) are found and their negative effects are mitigated at early stages of the software life cycle. With that in mind, architecture-based software performance optimisation (Aleti, A. et al., 2013; Rahmoun et al., 2017; Arcelli et al., 2018b) has been an active research topic among academia and industry in the field of software engineering and performance engineering. However, as software systems increase in size and complexity, the number of different architectural elements impacting system performance also grows. The values of these architectural elements can change within a certain range (e.g., the processing rate of a processor varies from 1MHz to 3MHz). The number of combinations of all possible values for all these architectural elements is potentially very large, i.e., the space of possible architectural choices impacting system performance is very large. More importantly, this space is intrinsically discontinuous, because other quality attributes (e.g., availability and cost) may be considered as constraints and the values of variable architectural elements can only be changed within a certain range. Manually searching for software architecture (SA) with good performance in such a large and discontinuous search space is time-consuming and costly. To increase the degree of automation of architecture-based software performance optimisation, diagnosis methods, rule-based and metaheuristic-based methods have been proposed on the basis of quality evaluation at the SA level in the past decades.

Diagnosis methods (Cortellessa et al., 2014; Trubiani et al., 2014; Sanctis, M. D. et al., 2017) can find potential performance problems and provide corresponding candidate improvement solutions according to performance antipatterns (Smith and Williams, 2003). Even though the performance problems and the candidate solutions can be automatically given by diagnosis tools, software architects still need to manually choose the best candidate solution and apply it to the SA model to be optimised. For more automation, rule-based and metaheuristic-based methods have been proposed to automatically optimise SA. In rule-based methods (Xu, 2012; Du et al., 2015b,a; Arcelli et al., 2018b,a), architecture-based software performance improvement knowledge from performance anti-patterns or design tactics (Koziolek et al., 2011a; Koziolek, 2014) can be formally described and automatically applied in the form of rules. Each rule contains precondition and action. The precondition is responsible to diagnose the performance problems by the predefined threshold, and the action is used to modify each element incurring problems by a certain improvement amplitude. Rule-based methods use optimisation algorithms (e.g., GA (Du et al., 2015b,a) or tree-based search strategies (Xu, 2012)) to search for the SAs with good performance by combining the predefined rules. In metaheuristic-based methods (Koziolek et al., 2013; Rahmoun et al., 2017; Du et al., 2017; Sedaghatbaf and Azgomi, 2019), the types of variable architectural elements (e.g., component allocation, hardware allocation, hardware configuration and component selection) are regarded as degrees of freedom (DoFs). Instead, the instances of DoFs (abbr. DoFIs) are treated as optimisation parameters and the ranges of the parameter values determine the search space. Accordingly, architecture-based performance optimisation is considered as a parameter optimisation problem. Some evolutionary algorithms (e.g. NSGA-II (Koziolek et al., 2013), (Rahmoun et al., 2017) and DE algorithm (Du et al., 2017)) have been proposed. Design strategies were further introduced to improve the solution quality and convergence rate (Koziolek, 2014). Metaheuristic-based methods have obtained good optimisation results even when taking other quality attributes (e.g., availability and cost) than performance into account.

As a new feature of architecture-based performance optimization method, explicability is discussed in our paper. For all kinds of methods, the optimisation re-

sults can be explained according to the differences between the initial SA and the resulting SA. The differences can be indicated by the modified SA elements. We claim that all methods have the trivial explicability (See Section 3.2.1). However, due to the application of the knowledge about architecture-based software performance improvement, diagnosis and rule-based methods can obtain optimisation results which can be used to explain how the resulting SA is obtained from the initial SA step by step. The cause of modification for SA can be known by stakeholders. We claim these methods have the full explicability (See Section 3.2.1).

Additionally, diagnosis methods, rule-based and metaheuristic-based methods also have several limitations:

(1) Diagnosis methods: although the optimisation process can be conducted semi-automatically by using the diagnosis tools iteratively, software architects still have the role of selecting good candidate improvement solutions and applying them. Manual selection and application of solutions is time-consuming, error-prone and non-trivial. And this has great effect on the performance improvements obtained in each iteration. Local optimal SAs may be obtained, especially when the constraints of multiple quality attributes and cost need to be compromised.

(2) Rule-based methods: in each rule, threshold in the precondition and improvement amplitude in the action need to be set before optimisation by software architects. Setting an appropriate threshold is rather hard because it needs a deep understanding of performance improvement knowledge and SA to be optimized in hand. At the same time, the explanation with threshold is not particularly convincing and reliable for architectural stakeholders. What's more, in each rule, the predefined threshold and improvement amplitude can potentially prevent optimisation algorithms from searching larger space. As a result, the better solutions may be excluded.

(3) Metaheuristic-based methods: most of these methods ignore the performance improvement knowledge represented by performance antipatterns and architectural tactics, and SA is randomly modified based on DoFIs to some extent. Therefore, it is difficult for stakeholders to understand the optimisation results and select SAs when trading off other quality attributes.

To address the above mentioned limitations, we propose a new approach to performance optimisation at the SA level based on randomised rules, named Multi-objective performance Optimisation based on Randomised search rulEs (MORE). MORE considers performance, availability and cost as different optimisation objectives and is composed of three novel components:

(1) Randomised search rules (MORE-R) are designed by introducing the maximum or minimum indices in precondition and randomizing modifications in action while also considering the other quality attributes (cost and availability). Unlike rules in the diagnosis methods and rule-based methods, MORE-R do not need to set any parameters (thresholds and improvement amplitudes) in rules and can improve representations of performance antipatterns or design tactics derived from practical experiences. Thus, MORE-R not only is easier to be used by software architects, but also provides explanation without parameters.

(2) An explicable multi-objective optimisation problem (MORE-P) is used to formulate architecture-based performance optimisation using MORE-R. In MORE-P, the explicability and its three metrics of $AvgNumMdf$, $avgNumRul_+$ and $avgNumMdf_+$ are first proposed. Furthermore, an explicable solution is defined as a variable-length sequence of rule applications satisfying some constraints, considering all possible composite applications of rules (e.g., the count, order and modified elements of each rule application). This can help to not only provide better explanations for optimisation results to architectural stakeholders, but also enlarge search space to obtain better solutions.

(3) A multi-objective evolutionary algorithm (MORE-EA) is designed to efficiently solve MORE-P. In MORE-EA, an explicable solution is represented as a fixed-length encoding by introducing a do-nothing

rule. The fixed-length encoding can reduce the complexity of evolutionary operators compared to a variable-length one. In addition, our evolutionary operators with repair mechanism can potentially improve the efficiency of MORE-EA and explanations for optimisation results.

We applied MORE to six problem instances with different scale and categories in order to compare with a representative metaheuristic-based method (PCM) and a typical rule-based method (PB). Our experimental results show that:

(1) Compared to PCM, MORE obtained explicable solutions of significantly better quality (p-value << 0.05 based on Wilcoxon rank-sum tests (Arcuri and Briand, 2011)) with high Vargha-Delaney $\hat{A}_{12}$ (Grissom and Kim, 2005) effect size in all compared experiments when optimising performance, cost and availability. At the same time, MORE obtained significantly better (p << 0.05) explanations for results in terms of the metric $AvgNumMdf$ (the average number of modifications to the initial SA), with very large Vargha-Delaney $\hat{A}_{12}$ effect size. The metric $AvgNumMdf$ was decreased by more than 17.5%, on all compared problem instances.

(2) MORE retrieves a set of explicable solutions with different trade-offs among quality metrics, whereas PB retrieves a single solution. On average, in all compared experiments, more than 40% of the solutions from MORE's set had significantly better response time and cost (p << 0.05) than PB's single retrieved solution, with very large Vargha-Delaney $\hat{A}_{12}$ effect size. Meanwhile, for the better solutions, MORE obtained significantly better (p << 0.05) explanations for results in terms of the two metrics $avgNumRul_+$ and $avgNumMdf_+$ (the average number of rule applications in the set of optimal solutions, and the average number of modifications in rule applications in the set of optimal solutions). The $avgNumRul_+$ and $avgNumMdf_+$ were improved by at least 33% and 35% respectively.

These experimental results show that MORE is able to achieve more explicable and higher quality solutions than PCM and PB. They also demonstrate the benefit of integrating search-based software engineering approaches with practical knowledge.

The rest of this paper is organised as follows. Section 2 introduces the background on architecture-based quality evaluation, architectural tactics for performance improvement and the methods compared against MORE. Section 3 presents our proposed approach MORE, including its randomised search rules, MORE-P and MORE-EA. Section 4 explains the design of our experiments. Section 5 presents our experimental results and analysis. Section 6 presents the threats to validity. Section 7 discusses the state-of-the-art in architecture-based performance optimisation and pinpoints shortcomings of current methods. Section 8 presents our conclusions and discusses opportunities for future work.

2. Background

In this section, architecture-based quality evaluation, architectural tactics for performance improvement and the compared methods will be introduced.

2.1. Architecture-based quality evaluation

A few architecture-based quality evaluation methods adopting well-known models and tools have been proposed in the past several years. These methods can be used during the process of optimising performance. In this section, we present the existing architecture-based quality evaluation methods with respect to performance, reliability and availability, and cost.

2.1.1. Performance evaluation

Performance evaluation is necessary for architecture-based software performance optimisation to acquire performance information. Architecture-based performance evaluation methods (Brosig et al., 2015) can automatically transform SAs, which are described in various languages or tools (e.g., UML2, AADL, PCM), into performance models, such as queueing network (Kounev, 2006), layered queueing network (LQN) (Tribastone, 2013), stochastic petri net (Distefano et al., 2011), stochastic process algebra

(Tribastone et al., 2012). The performance models are then used to create performance reports from which a few performance metrics (e.g., resource utilisation, response time or throughput) can be obtained. Therefore, architecture-based performance evaluation methods can support performance optimisation at the SA level to get necessary performance information.

Although different methods use different models, the core processes of performance evaluation are similar. First, a performance model (e.g., LQN) is derived from a SA model by use of a specific extractor. Then, a performance evaluation report is generated by using a model-related solver. Finally, the performance index values of software and hardware elements such as response time, throughput and resource utilisation, are acquired from the evaluation report. To compare with PCM and PB (Section 2.3), we take LQN as the performance model in this paper.

2.1.2. Reliability and availability evaluation

From an architecture point of view, reliability and availability are both operational qualities of a software system. Several measures are traditionally used for availability and reliability, such as mean time to failure, mean time to repair and failure rate (Immonen and Niemelä, 2008). Two categories of techniques have been proposed for architecture-based reliability and availability evaluation (Gokhale, 2007):

- Path-based approaches – in these approaches, several execution paths through the application are enumerated. Each path starts at the initial component and ends at the final component. The reliability of each path is obtained as a product of the reliabilities of the components along the path.
- State-based approaches – in these approaches, the probabilistic control flow graph of the application is mapped to a state space model. The state space model used to represent the SA can be a discrete-time Markov chain, a continuous time Markov chain or a semi-Markov process.

However, in most path-based and state-based approaches, the influence of system usage profile (i.e., sequences of system calls and values of parameters given as an input to these calls) on the control and data flow throughout the architecture is not explicitly modelled. The unavailability of the underlying hardware resources and communication failures across network links is usually not considered either. To eliminate these limitations, Brosch et al. (2012) proposed a novel reliability and availability evaluation approach based on the Palladio Component Model (PCM) (Becker et al., 2009). The approach (1) considers individual architectural impact factors on reliability, (2) explicitly models the component usage profile and execution environment, and (3) provides tools integrated into PCM platform to support the transformation of SA into a continuous time Markov chain and solve it to obtain reliability and availability indices. Koziolek's approach is adopted by our compared method (Koziolek et al., 2013) and also used by our approach to evaluate software availability through its architecture.

2.1.3. Cost evaluation

For architecture-based cost evaluation, development effort and delivery cost can be estimated from the structural, behavioural and deployment views in an early phase of software lifecycle by using methods such as COCOMO II (Boehm et al., 2009) and COSMIC (ISO, 2011). Machine learning approaches (Shepperd and Schofield, 1997; Minku and Yao, 2014; Sarro et al., 2016) could also potentially be used for that. A few architecture-based approaches (Poort and Vliet, 2015) have been proposed to estimate cost for complex industrial systems, which consist of multiple substantially different software, infrastructure and often organisational elements, integrated to deliver a coherent set of services. Some research (Slot, 2010) shows that using architectural information in cost estimation is correlated with higher accuracy of budget prediction. This is because, in addition to functionalities, quality attributes (i.e., performance, availability, etc.) are also considered. However, architecture-based performance optimisation approaches usually determine the trade-

off between cost and performance by using simplified cost estimation methods. To provide a fair comparison between existing work with ours, we adopt the same cost estimation method as the compared approaches in this paper.

2.2. Architectural tactics for performance improvement

Architecture-based performance knowledge accumulated by research and practice over the past decades has been systematically summarised in 12 performance anti-patterns (Navarro et al., 2013; Smith and Williams, 2003; Meier et al., 2004) and 17 performance improvement tactics (Koziolek et al., 2011a). These tactics integrate the anti-patterns with the performance improvement guidelines proposed by Microsoft (Koziolek et al., 2011a; Koziolek, 2014), and can be used to find potential performance problems and give best practice solutions.

Considering that SA usually describes the structure, behaviour and deployment of a system, we reorganise the tactics from Koziolek (2014) into three perspectives as shown in Figure 1. The 6 tactics in the deployment view involve the experience about the allocation of components and the capability of resources. The 6 tactics in the structure view are relevant to the knowledge which helps software architects to modify the structure of SA for better performance. The 5 tactics in the behaviour view can enhance performance by adjusting interaction behaviours, such as asynchronous communication, concurrency and parallelisation, and so on.

Furthermore, as mentioned in (Smith and Williams, 2003) and (Koziolek, 2014), software architects are encouraged to apply architectural tactics and remove anti-patterns to obtain better results in a process of architecture-based performance improvement. However, it is still a difficult problem how to combine architectural tactics automatically to obtain the best results. In this paper, we try to address this difficult problem.

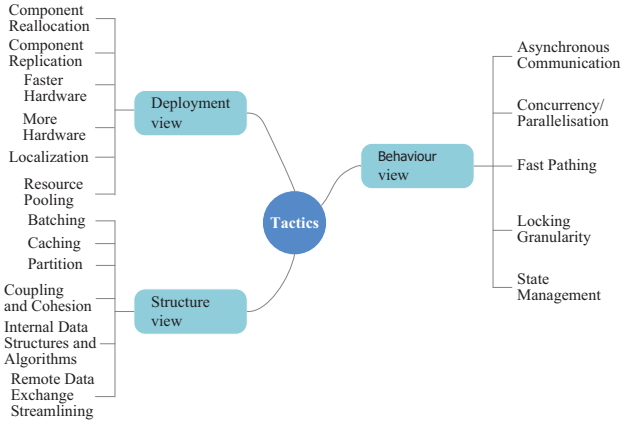


Figure 1: Architectural tactics for performance improvement

2.3. Overview of compared methods

In this section, we firstly give the reasons for selection of the compared methods. Then they are briefly introduced, respectively.

2.3.1. The reasons for selecting the compared methods

Among a variety of existing metaheuristic-based methods (See Section 7.3) and rule-based methods (See Section 7.2), the representative metaheuristic-based method (Koziolek et al., 2013) based on Palladio Component Model and the typical rule-based method (Xu, 2012) based on the tool named Performance Booster (PB) are selected as the compared methods. In this paper, the compared methods are abbreviated as PCM and PB, respectively. The reasons for selecting them are given below:

(1) PB and PCM methods use Unified Modeling Language (UML) and Palladio Component Model to describe SA, respectively. UML is a de facto standard modeling language in the field of software engineering. While Palladio Component Model is widely applied and has become a commercial tool.

(2) Most of the problem instances used by PB and PCM are from real world applications or systems of different categories. And their complexity and scale are different. As a result, this can guarantee the diversity of problem instances in order to obtain the convincing experimental results. More details can be found in Table 7.

(3) PB (Xu, 2012) and PCM (Koziolek et al., 2013;

Koziolek, 2014) methods outperform other methods in their respective categories. Furthermore, both of them balanced between performance and other quality attributes (such as availability and cost) during optimisation.

2.3.2. PCM method

PCM method (Koziolek et al., 2013) can optimise QoS of component-based software systems with the objectives of minimising response time, maximising availability and minimising cost. It considers three types of DoFs- allocation of components, server configuration and component selection. The different possible values for these DoFs form the search space. The optimisation process is divided into analysis optimisation and evolutionary optimisation. The analysis optimisation takes a component-based architecture specified by a PCM instance as an initial candidate and optimises it based on a simplified procedure. This simplified procedure investigates a limited search space and simplified evaluation models, but is fast to run. It results in a set of different architectures with different trade-offs between the three different objectives. This set is then used as candidate solutions when initialising a multi-objective evolutionary algorithm, which further optimises the architectures denoted by candidate solutions through investigating a larger search space. The evolutionary algorithm uses LQN for deriving the response time, discrete-time Markov chain for obtaining availability, and predefined cost model to evaluate the total cost. In this model, both hardware resources and software components are needed to annotate their costs of ownership and these costs were added together to form the total cost. Especially, the hardware resource cost is in proportion to the processing rate of processor. Furthermore, if a server specified in the model is not used, i.e. no components are allocated to it, its cost is not added to the overall cost.

2.3.3. PB method

PB (Xu, 2012) is a rule-based automated software performance optimisation method based on SA models described by UML with Schedulability, Performance and Time (SPT) profiles (OMG, 2017). In PB method, a few rules for performance optimization are defined based on performance antipatterns. PB can help to find and remove the performance flaws (i.e., bottlenecks and long execution paths) at the level of LQN model. In addition, PB method adopted tree-based search optimisation algorithms. Meanwhile, the cost is regarded as a design constraint with respect to performance optimisation. The performance improvement rules, performance optimisation algorithm and calculation of cost used by PB method are briefly explained below.

(1) Performance improvement rules

Two categories of rules are defined based on LQN model. The first category serves performance diagnosis and improvement for the LQN model. The second is used to describe how the changes in LQN model are interpreted as changes in the UML design model with SPT annotation. As the second category of rules is used only for synchronising the LQN and UML models, we discuss only the first category in this paper.

Each rule contains precondition and action. For example, the rule "More hardware" is stated as "If the utilisation rate U_p of processor P is greater than a threshold U_{sat}, then P's multiplicity is set to M_p.". And M_p is equal to $(1 + \Delta) * M'_p$ and not greater than $M_{\max,p}$, where M'_p is the original multiplicity of P; Δ and $M_{\max,p}$ represent improvement amplitude needed to set and the upper bound of P's multiplicity, respectively. Based on U_{sat} in the precondition, the processors that have bottleneck problem can be found. Meanwhile, the action modifies P's multiplicity. From the example, it can be seen that the threshold U_{sat} and the improvement amplitude Δ can influence the result of application of the rule "More hardware". However, they need to be set by software architects before optimisation, which is a non-trivial task because it needs a deep understanding of performance improvement knowledge and SA to be optimized in hand.

In addition, the performance improvement rules, which are related to seven DoFs (redeployment, multiplicity of processor, multiplicity of software component, call across network, partition, demand for proces-

sor and asynchronous communication) for LQN model, are divided into configuration improvement rules and design improvement rules. The former rules are used to find the bottleneck problems caused by unreasonable utilisation or allocation of resources and eliminate these problems by means of resource reconfiguration. The latter rules are used to discover long path problems (longer response time on the execution or internal path than the required one) caused by inappropriate designs.

(2) Optimisation algorithm

The optimisation algorithm uses the above mentioned performance improvement rules and adopts tree-based strategy to search the best solution. In Figure 2, r_1, r_2 and r_3 are the configuration improvement rules. And $r_4, r_5, ..., r_7$ represent the design improvement rules, respectively. The tree-based strategy is described as a search tree and divided into several search rounds. It starts from the root node LQN_0 extracted from the initial software architecture SA_0. Firstly, in the configuration space, the sequence of configuration improvement rules is repeatedly applied to LQN_0, until no further performance improvement is obtained. At this time, a candidate model can be obtained. Then, a search round for this candidate model starts. In Figure 2, the dashed rectangle represents a search round.

In a search round, each of design improvement rules is applied in parallel to this model, resulting in a set of improved models. Each improved model is further optimised by repeatedly applying the sequence of configuration improvement rules. So, a set of candidate models is obtained and then evaluated in the end of search round. There are two strategies to decide which model will be selected to be the start model for the next search round. One is cost first and the other is response time first. In cost first, the candidate model with the minimum accumulative cost from LQN_0 to the current model is selected. In response time first, the candidate model with the minimum response time is selected. The whole search process terminates when the performance indices satisfy the requirements or cannot be improved. The PB methods with cost-first and response time-first will be denoted by PB_{cost} and PB_{time} in this paper, respectively.

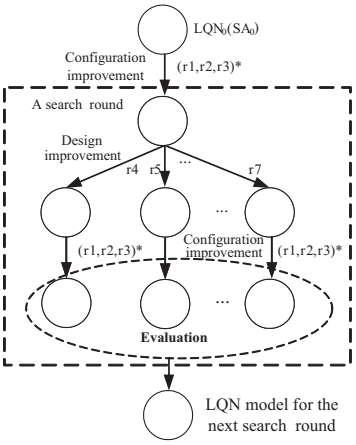


Figure 2: PB method's search process

(3) Cost estimation

The cost of configuration improvements is defined as zero, whereas the cost of design improvements and their corresponding implementation effort is computed as a proportion of the improvements obtained for LQN elements. The motivation for considering cost as a proportion of the improvements is illustrated with the following example. Consider that a service is designed to respond to a request in 1 ms. In order to reduce the response time by 20%, a more efficient algorithm may need to be selected and much greater cost may need to be paid than for reducing the response time by 10%.

The cost of design improvements and their implementation effort consider three factors. First, the action in each design improvement rule is defined to modify problematic elements by the predefined proportion of the initial value. The larger the predefined proportion, the larger the cost. Second, there are possibly different rules to improve a given performance problem (e.g., long path). Different rules may require different efforts, which are represented by the weights given by the PB method. Third, when a design improvement rule is applied to different LQN models, the number of modified elements may be different. The more elements are modified, the larger the cost. A more formal description of cost computation in the PB method is given below.

The improvement cost of the *root* node equals to

zero (Eq.(1)), because none of the rules has been used.

$$cost(root) = 0 \tag{1}$$

For a non-root node $node_i$, the cost is calculated from its parent node using Eq.(2):

$$cost(node_i) = \sum_{k=1}^{n} w_{r_j} * ampl_k \tag{2}$$

where n is the number of LQN elements of $node_i$'s parent node that were modified to generate $node_i$ using a given rule r_j; $ampl_k$ is the improvement amplitude of the k^{th} LQN element; and w_{r_j} represents a unit cost and is defined by the architects when the rule r_j is executed and SA is modified by a unit of improvement amplitude.

The improvement cost of path l from the root node is defined using Eq.(3). It is regarded as an accumulative cost and used by PB method.

$$cost(l) = \sum_{node_i \in l} cost(node_i) \tag{3}$$

3. Our proposed approach MORE

This section presents our approach MORE, including randomised search rules (MORE-R, Section 3.1), MORE optimisation problem (MORE-P, Section 3.2) and its solving algorithm (MORE-EA, Section 3.3).

3.1. Randomised search rules (MORE-R)

In this subsection, definition of randomised search rules is presented firstly. Then selection and addition of randomised search rules when considering other quality attributes are given. Finally, two examples of selection and addition of rules for MORE are shown.

3.1.1. Definition of randomised search rules for architecture-based performance improvement

Based on architectural tactics for performance improvement in Figure 1, randomised search rules are designed and shown in Tables 1, 2 and 3 where each rule identified by a number contains precondition and action, and the last column indicates whether the rules need performance information from a performance evaluation report.

To alleviate the problems incurred by threshold in the precondition and improvement amplitude in the action in rule-based methods (see Section 2.3.3), the maximum or minimum indices and randomised modifications are introduced to define precondition and action of a randomised search rule, respectively. These indices are related to design elements (shown in Rule 7, Rule 8, Rule 12) and performance information (shown in Rule 1, Rule 2, Rule 3, Rule 4, Rule 5, Rule 6, Rule 9, Rule 11, Rule 14, Rule 16). They are automatically set by analyzing the SA to be optimized or checking its performance evaluation report. For example, the precondition of Rule 1 is responsible for picking up the processor P with highest utilization in a SA by checking the performance evaluation report. The precondition of Rule 7 is used to locate the call cal whose number is the maximum among all calls across the network in a SA by analyzing the structure of the SA. The precondition is used to locate the architectural element incurring performance problems based on the current SA and its performance information. Instead, the action can randomly generate modifications to be applied to the current SA if the precondition is satisfied.

These rules in MORE are organised by three views: deployment, structure and behaviour. Table 1 shows the deployment rules. They can address hardware and software resource bottlenecks by randomly reallocating resources, adding multiplicity or instances of resources, increasing the processing rate of resources, etc. Table 2 shows the structure rules. These rules can be applied to improve utilisation, throughput and response time by randomly changing the structural elements of SA, such as the port of component, connection and responsibility assignment. Table 3 shows the behaviour rules. They can reduce response time by randomly modifying behaviour elements, which can introduce more asynchronous communication, concurrency and parallelisation, etc.

Table 1
Randomised search rules in deployment view. Column 'requires performance report' indicates whether the rule needs a performance evaluation report to decide whether its conditions are satisfied.

No.	Tactic Name	Randomised search rule		Anti-patterns Relieved by the Rule	Requires Performance Evaluation Report
		condition	action		
Rule 1	Component Reallocation	Processor P has the highest utilisation rate and a set of components $Coms$ are deployed on P.	Select a component from $Coms$ at random and deploy it on the processor with lowest utilisation rate.	Extensive Processing	Yes
Rule 2	Component Replication	Component C not only has the highest utilisation rate but also is the most accessed by other components.	Add new instances of C at random, respecting an upper bound on the number of instances of C, and randomly deploy these new instances on available servers.	One-Lane Bridge	Yes
Rule 3	Faster Hardware	Processor P has the highest utilisation rate.	Increase the processing rate of P randomly, respecting an upper bound on the processing rate of P.	Extensive Processing	Yes
Rule 4	More Hardware	Processor P has the highest utilisation rate.	Increase the multiplicity of P randomly, respecting an upper bound on the multiplicity of P.	One-Lane Bridge	Yes
Rule 5	Localisation	Component $C1$ has the highest utilisation rate and interacts with a set of component $Coms$.	Randomly select a component $C2$ from $Coms$ and redeploy $C1$ on the processor where $C2$ is.	Extensive Processing	Yes
Rule 6	Resource Pooling	Passive resource R with the highest waiting delay.	Increase the capacity of R randomly, respecting an upper bound on the capacity of R.	One-Lane Bridge	Yes

3.1.2. Selection and addition of randomised search rules when considering other quality attributes

(1) Selection of rules

Selection of randomised search rules depends on Architectural Description Language (ADL) used to describe SA. Different ADLs have different syntax to describe SA with respect to structure, behaviour and deployment at different abstract levels. Furthermore, for the SA described in a specific ADL, only a few types of variable architectural elements can be defined as DoFs for performance improvement. Therefore, only a subset of the 17 randomised search rules can be selected due to limited description capability of an ADL and the permitted DoFs in the process of performance optimisation. Specifically, from Tables 1, 2 and 3, we select the rules whose actions are dependent of the permitted DoFs. Additionally, it's worth noting that one DoF can correspond to multiple rules.

(2) Addition of rules

The selected rules might potentially influence other quality attributes such as availability and cost. In order to enable the exploration of more trade-offs between performance and other quality attributes, the additional randomised search rules may be designed to improve other quality attributes and degrade performance. There are two cases for designing the additional rules.

One case is for the negative influence on other quality attributes incurred by the action of a selected rule in the existence of empty servers in SA. By investigating 17 randomised search rules, we find that only Rule 1 falls into this case. Rule 1 improves performance by reallocating a component from the server with the highest utilisation rate to the server with the lowest utilisation rate. When Rule 1 is executed and the empty servers exist in SA, a component must be reallocated to one of empty servers because the empty servers have the lowest utilisation. An empty server becomes the server that holds only one component. If this definitely degrades one of other quality attributes, it is necessary to design an additional rule to redeploy the component from this server to a permitted non-empty server so that this server becomes an empty server again. The additional rule is designed as Rule 18 shown in Table 4. Another case is when a selected rule whose action can deterministically degrade one of other quality attributes, an

Table 2
Randomised search rules in structure view. Column 'requires performance report' indicates whether the rule needs a performance evaluation report to decide whether its conditions are satisfied.

No.	Tactic Name	Randomised search rule		Anti-patterns Relieved by the Rule	Requires Performance Evaluation Report
		condition	action		
Rule 7	Batching	Cal is a call across the network. The number of Cal is the maximum among all calls across network and is greater than 1.	Randomly reduce the number of Cal and increase the size of the data transmitted in Cal.	Empty Semi Trucks	No
Rule 8	Caching	Cal is a call. The number of Cal is the maximum among all calls and is greater than 1.	Create an arbitrary number of cache components for Cal.	Extensive Processing and Circuitous Treasure Hunt	No
Rule 9	Partition	Component $C1$ has the highest utilisation rate.	Create a component $C2$ and randomly move several operations from $C1$ into $C2$, and these operations don't interact with the remaining operations in $C1$.	God Class/Component and Extensive Processing	Yes
Rule 10	Coupling and Cohesion	$Cals$ is a set of calls and the number of each call in $Cals$ is greater than 1.	Select a call Cal from $Cals$ randomly and merge the two components which are the caller and callee of Cal.	Empty Semi Trucks	No
Rule 11	Internal Data Structures and Algorithms	Component C has the highest utilisation rate and there is a set of components $Coms$ that can replace C	Select a component $C2$ from $Coms$ randomly and replace C with $C2$.	Extensive Processing	Yes
Rule 12	Remote Data Exchange Streamlining	Cal is a call across network and the size of transferred data by Cal is the largest.	Create two components $C1$ and $C2$ to compress and decompress data, respectively. The compression rate of component $C1$ is set arbitrarily. $C1$ and $C2$ are used for the caller and callee of Cal.	Empty Semi Trucks	No

additional rule must be designed by defining its precondition and action that are opposite to the counterparts of this selected rule, as mentioned by (Koziolek et al., 2011a).

3.1.3. Examples of selection and addition of randomised search rules for MORE

Selection and addition of rules for MORE are influenced by four factors, including ADL, DoFs, other quality attributes and their own evaluation methods. However, it is impossible to demonstrate all possible combinations of four factors. To serve the comparative experiments (See Section 4.4), we choose PCM and PB as two references to decide the four factors for MORE. Therefore, we consider the same four factors as the referenced method.

(1) Selection and addition of rules for MORE when using PCM as a reference

In PCM, allocation of components (AC), server configuration (SC) and component selection (CS) have been considered as three types of DoFs. Correspondingly, we find that Rule 1 and Rule 5 are related to AC, and Rule 3 is relevant to SC. The three rules can be selected from Table 1. Moreover, Rule 11 associated to CS can be selected from Table 2.

Based on the two cases for designing additional rules in Section 3.1.2, we need to design additional rules for Rule 1 and Rule 3, respectively, after investigating the four selected rules. According to the first case, the action of Rule 1 makes an empty server become a non-empty server. This increases cost because the cost of the non-empty server must be added to the overall cost based on cost evaluation used in PCM (See Section 2.3.2). So, Rule 18 can be used to reduce cost because this non-empty server becomes an empty server. For the second case, Rule 19 in Table 4 is designed for Rule 3. Specifically, the precondition of Rule 3 is used to find the processor with the highest utilisation rate, and the action of Rule 3 is responsible to randomly increase processing rate of the processor. Based on cost evaluation used in PCM (See Section 2.3.2), cost must rise as the processing rate of processor in-

Table 3
Randomised search rules in behaviour view. Column 'requires performance report' indicates whether the rule needs a performance evaluation report to decide whether its conditions are satisfied.

No.	Tactic Name	Randomised search rule		Anti-patterns Relieved by the Rule	Requires Performance Evaluation Report
		condition	action		
Rule 13	Asynchronous Communication	The response time of service Srv is greater than an expected value and there is a set of synchronous calls $Cals$ that can be changed into asynchronous calls in Srv's call path.	Select one or more calls from $Cals$ randomly and change them into asynchronous calls.	Concurrent Processing Systems	Yes
Rule 14	Concurrency Parallelisation	Component C has the highest utilisation rate and has a thread pool.	Increase the size of C's thread pool randomly, respecting an upper bound.	Concurrent Processing Systems	Yes
Rule 15	Fast Pathing	The response time of service Srv is greater than an expected value and Srv contains multiple execution paths denoted by $Paths$.	Randomly select a path whose response time is not the longest in $Paths$, and introduce additional components for this path.	Extensive Processing	Yes
Rule 16	Locking Granularity	Component C has the highest utilisation rate and there is a set of actions Acs which can reduce the time of acquiring and releasing lock in C's internal behaviours.	Select several actions from Acs randomly and modify the behaviours of acquiring and releasing lock in them.	One-Lane Bridge	Yes
Rule 17	State Management	Component $C1$ is a state component and there is a group of components $Coms$ that can replace $C1$.	Randomly select a component $C2$ from $Coms$ and replace $C1$ with $C2$.	One-Lane Bridge and Excessive Dynamic Allocation	No

creases. Therefore, Rule 19 decreases processing rate of processor with the lowest utilisation rate because we expect that costs are saved, while performance is degraded as slightly as possible.

Table 4
The additional randomised search rules for MORE when using PCM as a reference

No.	Randomised search rule	
	condition	action
Rule 18	Only one component C is deployed on Server S	Randomly redeploy C from S to one of permitted non-empty servers
Rule 19	Processor P has the lowest utilisation rate	Decrease the processing rate of P randomly, respecting a lower bound on the processing rate of P

(2) Selection and addition of rules for MORE when using PB as a reference

In PB, redeployment, multiplicity of processor, multiplicity of software component, call across network, partition, demand for processor and asynchronous communication have been considered as seven types of DoFs. Thus, based on the seven DoFs, Rule 1, Rule 4, Rule 6, Rule 7, Rule 9, Rule 11 and Rule 13 are selected from Tables 1, 2 and 3 correspondingly.

No additional rules are designed for MORE when PB is used as a reference. The reasons are given as follows. Only the tradeoff between performance and cost should be considered in MORE because the annotation related to availability has not been specified in PB. Cost in PB is evaluated as an accumulative cost and is based on the use of each rule in the process of optimisation. However, we cannot predict which of the seven selected rules will be used during optimisation and there is no deterministic degradation of cost incurred by these selected rules. Moreover, although Rule 1 is selected for MORE, cost evaluation of PB does not consider hardware resource cost so that Rule 18 is not regarded as an additional rule.

3.2. The MORE optimisation problem (MORE-P)

In this subsection, the related concepts to explicability are introduced. And based on these concepts, we give the novel formulation of architecture-based performance optimisation using randomised search rules.

3.2.1. The related concepts to explicability

(1) The explicability

Explicability is a new feature of architecture-based performance optimization method proposed in our pa-

per. It is used to characterize the ability of optimization method to provide feedback to stakeholders. All kinds of methods have the trivial explicability because the optimisation results can be explained according to the difference between the initial SA and the resulting SA. Instead, rule-based methods have the full explicability because performance improvement knowledge can be applied to explain the optimisation results to stakeholders. Consequently, the stakeholders can understand how the resulting SA is obtained from the initial SA step by step.

The reason why rule-based methods have the full explicability is illustrated as follows.

- Rules are used to represent performance improvement knowledge which can find potential performance problems and give best practice solutions. The precondition of rule is responsible to diagnose the performance problems in a context including a SA and its performance information, while the action of rule is used to modify SA elements incurring problems based on best practice solutions.

- A rule application can be used to explicitly represent that a rule is applied once in a context. It potentially includes the applied rule, its execution context, and the corresponding modifications to SA elements. So, a rule application not only indicates why to modify SA and which architectural elements need to be modified by the precondition of rule, but also decides the results of modifications by the action of rule.

- Any process of obtaining the resulting SA from the initial SA can be regarded as continuous applications of rules. Naturally, these continuous applications can be properly represented by a sequence of rule applications when the context of each rule application in this sequence is correctly built based on its precursor one, except for the first rule application. After an optimisation result is represented as a sequence of rule applications, it can be well explained to the architectural stakeholders step by step.

Instead, metaheuristic-based methods, as representative black-box methods, have only the trivial explicability, due to absence of explanation for causes of modifications.

(2) The metrics of explicability

In our paper, explicability is embodied by explanations for optimisation results. Based on optimisation results, different metrics, $AvgNumMdf$, $avgNumRul_+$ and $avgNumMdf_+$, are designed to achieve comparison on explicability between different methods. The metric $AvgNumMdf$, which represents the number of modifications to the initial SA in the set of optimal solutions, is proposed to measure trivial explicability. So, this metric is used for comparison between a rule-based method and a metaheuristic-based method, or between two metaheuristic-based methods. The metrics of $avgNumRul_+$ and $avgNumMdf_+$ are designed to evaluate the number of rule applications and the number of modifications in rule applications in the set of optimal solutions, respectively. The two metrics are both used for comparison between two rule-based methods because the trivial and the full explicability should be considered together.

Meanwhile, the two metrics of $avgNumRul_+$ and $avgNumMdf_+$ are encouraged to be used together in order to avoid two situations that have potentially negative influence on the explainations for the results. In the first situation, when $avgNumRul_+$ is regarded as sole metric, the method with smaller value of $avgNumRul_+$ is better. This means the results obtained by this method include fewer rule applications. However, it is possible that more modifications are generated by these rule applications. In the second situation, given that $avgNumRul_+$ is sole metric, the method with smaller value of $avgNumMdf_+$ is better, but the results obtained by this method possibly include more rule applications.

The three designed metrics are shown in Table 5, where rule-based method and metaheuristic-based method are denoted by RM and MM, respectively.

Considering that the set of Pareto optimal solutions may be acquired by the compared methods, we adopt arithmetic mean in the definitions of these metrics. The smaller the values of these metrics, the better the explanations for results.

Table 5
Three metrics of explicability in comparison of different methods based on their optimisation results (RM: rule-based method, MM: metaheuristic-based method)

Metric	Description	Comparison of methods
$AvgNumMdf$	the average number of modifications to the initial SA in the set of optimal solutions	RM vs MM or MM vs MM
$avgNumRul_+$	the average number of rule applications in the set of optimal solutions	RM vs RM
$avgNumMdf_+$	the average number of modifications in rule applications in the set of optimal solutions	RM vs RM

3.2.2. The formulation of MORE-P

Based on the explicablity and its metrics, formulation of MORE-P will be presented. Notations used by MORE-P are given in Table 6.

Definition 1. (Rule Application) *A rule application, shorted by rulApp, is represented by a 3-tuple (r, ctx, E) where r, ctx and E are a rule number, an execution context and the modifications when applying rule r in the context ctx.*

For ease of description, $rulApp[r]$, $rulApp[ctx]$ and $rulApp[E]$ represent the rule number, the context and the modification of $rulApp$, respectively. Furthermore, the function $exe(r, ctx)$ is used to define the returned modifications E when applying rule r in the context ctx. As a result, a rule application $rulApp$ not only indicates why to modify SA and which architectural elements need to be modified by the precondition of rule $rulApp[r]$, but also decides the result of modification E by randomised action of rule $rulApp[r]$ in its context $rulApp[ctx]$.

Table 6
Notations used by MORE-P

Symbol	Description
SA_0	The initial SA.
n	The number of randomised search rules.
r	r is the rule number ($1 \leqslant r \leqslant n$).
E	Modifications are defined as a set of id-value pair which describe the ID and new value of a modified element (a DoF instance), respectively.
$ids(E)$	Function that returns the set of IDs which occur in E.
ctx	An execution context of a randomised search rule, which comprises a SA and its performance information. ctx_1 is defined as the initial context which is made of SA_0 and its performance information. $ctx[SA]$ represents the SA taken from ctx .
$exe(r, ctx)$	Function that returns the modifications E when applying rule r in the context ctx.
$bldCtx(E, ctx)$	Function that builds a new context by two steps: First, modification E is applied into $ctx[SA]$ and a new SA can be obtained; Second, the performance information corresponding to this new SA can be acquired by performance evaluation, and then the new context is built.
m	The number of quality indicators considered in the architecture-based software performance optimisation problem. In this paper, three quality indicators (response time, 1-availability and cost) are considered.
$qIdx_j(SA)$	Evaluation function returns the value of the $j^{th}(1 \leq j \leq m)$ quality indicator of SA. For convenience and without loss of generality, we define that the smaller the value of $qIdx_j$, the better the quality indicator.

Definition 2. (Explicable Solution) *As shown in Eq.(4), an explicable solution X for performance optimisation problem based on randomised search rules is defined as a sequence of rule applications, satisfying the constraints on length, contexts, and explanations shown in Eq.(5), Eq.(6) and Eq.(7), respectively. In Eq.(4), x_k represents the k^{th} rule application in X.*

$$X =< x_1, \cdots, x_k, \cdots, x_l >, \tag{4}$$

$$\begin{aligned} l \leq s = \sum_{i=1}^{n} u_i, \\ 0 \leq h_i(X) \leq u_i \quad (1 \leq i \leq n) \end{aligned} \tag{5}$$

$$\begin{aligned} x_1[ctx] = ctx_1 \ \wedge \ E_k = exe(x_k[r], ctx_k) \wedge \\ x_{k+1}[ctx] = bldCtx(E_k, ctx_k), \ (1 \leq k \leq l-1) \end{aligned} \tag{6}$$

$$\begin{aligned} (\forall i, x_i[E]) \neq \phi \ \wedge \ 1 \leq i \leq l) \ \wedge \ (\forall k, 2 \leq k \leq l \wedge \\ x_{k-1}[r] = x_k[r] \longrightarrow ids(x_{k-1}[E]) \neq ids(x_k[E])) \end{aligned} \tag{7}$$

(1) The constraints on length

The length of an explicable solution X is equal to

the number of rule applications in X. Thus, to set the maximum length s of X, architects have to consider the number of applications of each selected rule. If u_i represents the maximum allowed number of applications of selected rule i, s is equal to the sum of u_i as shown in Eq.(5) where n is the number of selected rules. At the same time, the actual number of applications of rule i in X, denoted by h_i, should be not greater than u_i.

It is very important to set a reasonable value to u_i ($1 \leq i \leq n$) because the results and elapsed time of optimisation will be appreciably affected. If u_i is too large, MORE has to find solutions in a very huge search space and the whole optimization becomes too time-consuming. If u_i is too small, the size of search space becomes very small and the good solutions are possibly excluded. In order to alleviate this problem, a heuristic approach to setting u_i is given. Specifically, rule i is related to its DoFs (the types of variable architectural elements). The instances of these DoFs, shorted by DoFIs, can be often determined by the initial SA and its constraints, and potentially have equal chances to be modified by the randomised action of rule i. Thus, the number of DoFIs can be regarded as the heuristic information on setting u_i. Section 4.4 gives an example for setting u_i.

(2) The constraints on contexts

In Eq.(6), the context of each rule application x_k in an explicable solution X is determined by its precursor one, except for the first rule application. Specifically, ctx_1 is the first context including SA_0 and its performance information. Since it is based on the context $x_k[ctx](1 \leq k \leq l-1)$, the context of $x_{k+1}[ctx]$ should be built by the function $bldCtx$ through two steps. Firstly, modification E_k, got by executing the rule $x_k[r]$ in the context ctx_k, is applied into the $ctx_k[SA]$, and $ctx_{k+1}[SA]$ can be obtained. Secondly, performance information corresponding to $ctx_{k+1}[SA]$ can be acquired by performance evaluation, and then ctx_{k+1} is built.

(3) The constraints on explanations

To obtain better explanations for results in terms of the metrics of $AvgNumMdf$, $avgNumRul_+$ and $avgNumMdf_+$, an explicable solution X should satisfy the constraints defined by Eq.(7), which is a conjunctive formula and includes two predicates. The two predicates are used to assert that there are no useless and redundant rule applications in X, respectively. Specifically, in Eq.(7), ϕ is the empty set, $x_k[r]$ and $x_k[E]$ are the rule number and modifications of the k^{th} rule application in X, respectively. While $ids(x_k[E])$ represents the IDs of modified elements E of the k^{th} rule application in X. The two predicates are elaborated as follows.

- For any rule application x_i in X, if the modifications $x_i[E]$ is empty set when applying rule $x_i[r]$ in the context $x_i[ctx]$, x_i is regarded as a useless rule application in the context $x_i[ctx]$.
- For any two adjacent rule applications in X, x_{k-1} and x_k, if the rule numbers, $x_{k-1}[r]$ and $x_k[r]$, are same and IDs of modified elements, denoted by $ids(x_{k-1}[E])$ and $ids(x_k[E])$, are also same, this means that the same rule was sequentially applied twice, and the same elements were modified. In this case, the rule application x_{k-1} is regarded as a redundant rule application.

Definition 3. (Domination) *An l_1-length explicable solution X_1 dominates (Srinivas and Deb, 1994) another l_2-length explicable solution X_2 if and only if the condition defined by Eq.(8) is satisfied.*

$$\begin{aligned} &\forall j, qIdx_j(rltSA_1) \leq qIdx_j(rltSA_2) \wedge \\ &\exists k, qIdx_k(rltSA_1) < qIdx_k(rltSA_2), \\ &where\ j,k \in \{1,2,\cdots,m\}, \\ &rltSA_1 = ctx_{l_1+1}[SA] \\ &rltSA_2 = ctx_{l_2+1}[SA]. \end{aligned} \tag{8}$$

In Eq.(8), $qIdx_j$ is the evaluation function which returns the j^{th} quality index of the inputted SA. Two resulting SAs of $rltSA_1$ and $rltSA_2$ are taken out from ctx_{l_1+1} and ctx_{l_2+1} obtained by the function $bldCtx$ with the modifications and contexts in the last rule application, respectively.

Definition 4. (Explicable Pareto Optimal Solutions) *The set of explicable Pareto optimal solutions, Opt^*, is defined as Eq.(9).*

$$Opt^* = \{X | X \text{ is an explicable solution and non} - \text{dominated by any other explicable solutions in the search space}\} \quad (9)$$

Definition 5. (Architecture-based Performance Optimisation Using Randomised Search Rules) *This is a multi-objective optimisation problem which consists in finding the explicable Pareto optimal solutions defined by Eq.(9).*

3.3. The MORE multi-objective evolutionary algorithm (MORE-EA)

To solve MORE-P defined in the above section, a multi-objective evolutionary algorithm named MORE-EA is proposed based on NSGA-II (Deb et al., 2002). NSGA-II, a classic multi-objective algorithm, has been cited more than 30000 times from Google Scholar. It has been applied to many real-world and difficult multi-objective optimization problems (Deb, 2011), and has found near the true Pareto optimal solutions.

MORE-EA's main framework shown in Algorithm 1 is similar to NSGA-II's. However, there are some differences between them in the four steps: encoding (line 1), decoding (lines 4-6), evolutionary operations with repair mechanism (line 19) and the postprocessing step (lines 23-26). The last step is responsible to generate the set of explicable Pareto optimal solutions from the set of Pareto optimal encodings based on Eq.(7). While the first three steps will be presented as follows.

3.3.1. Encoding

In this section, a novel fixed-length encoding is introduced firstly. Then its example is given. Finally, the generation of random individuals in the initial population is presented.

(1) A fixed-length encoding

A candidate explicable solution can be represented as a fixed-length encoding by introducing a do-nothing rule. The do-nothing rule is numbered by 0 and does not modify any architecture elements when executed in any context. To represent the 0-length explicable solution (see an example of fixed-length encoding in this section), the do-nothing rule can be used s times at most where s is equal to the maximum possible length of any explicable solution, as defined by Eq.(5).

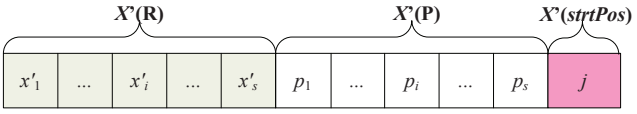


Figure 3: An encoding X'

An encoding X' has three segments, as shown in Figure 3. The first segment $X'(R)$ with gray background is used to store the integer rule numbers $0 \leq x'_i \leq n$ and has size s. The second segment $X'(P)$ with white background represents the pointers to the modified elements and has also size s. After the rule x'_i denoted by the segment $X'(R)$ is executed, the corresponding modifications $E_{x'_i}$ are referred by p_i in the segment $X'(P)$. The last segment X'(strtPos) with pink background is a single element used to determine where to start executing the sequence of rules denoted by $X'(R)$ in order to save the computation cost of rule execution by avoiding recomputation of $E_{x'_i}$($1 \leq i < X'(strtPos)$) when $X'(strtPos) > 1$. Therefore, if all rules specified by $X'(R)$ have been executed, $X'(strtPos)$ will be set as $s + 1$. So, the range of $X'(strtPos)$ is between 1 and $s + 1$. From Figure 3, we see the length of an encoding X' is equal to $2s + 1$.

(2) An example of a fixed-length encoding

Any variable-length explicable solution X with the maximum length s can be represented as a $(2s + 1)$-length encoding by the following four steps: Firstly, rule numbers in all rule applications of X are sequentially stored on the elements in $X'(R)$. Secondly, all the elements in $X'(P)$ are filled with the pointers to empty set. Thirdly, $X'(strtPos)$ is set as 1. Fourthly, if the length of X is l and $l < s$, the last $(s - l)$ elements in $X'(R)$ are respectively filled with 0. The filled 0 in $X'(R)$ are related to the do-nothing rules, and they don't change the meanings of X.

An example is shown in Figure 4 where s and l

Algorithm 1: MORE-EA

Input: The initial SA SA_0, population size N_p, mutation rate p_m, crossover rate p_c
Output: The set of explicable Pareto optimal solutions
$g = 0$, $Initialise(P_g)$, $Q_g = \phi$; // P_g and Q_g are the g^{th} population and its offspring population, respectively
while (termination condition is not met) **do**
  // Merge the parent and its offspring population
  $T_g = P_g \bigcup Q_g$;
  for (each encoding X' in T_g) **do**
    decode(X', SA_0); //decode X' based on SA_0, and acquire the corresponding objective values (i.e. quality //indices).
  end for
  $F = NondominatedSort(T_g)$; // Get all nondominated fronts of T_g by the nondominated sort according to the // objective values of each individual in T_g, and store those fronts in order of //ranking in $F = \{F_1, F_2, ...\}$
  $P_{g+1} = \phi, i = 1$; // Initialise the next parent population
  while ($(|P_{g+1}| < N_p) \wedge (|P_{g+1}| + |F_i| \leqslant N_p)$) **do**
    $P_{g+1} = P_{g+1} \cup F_i$; // Put the individuals of front F_i into the population P_{g+1}
    $i = i + 1$;
  end while
  if ($(|P_{g+1}| < N_p) \wedge (|P_{g+1}| + |F_i| > N_p)$) **then**
    calculateCrowdingDistance(F_i); // Compute the crowding distance in front F_i
    sortByCrowdingDistance(F_i); // Sort the individuals in front F_i by descending order based on // crowding distance
    $P_{g+1} = P_{g+1} \cup F_i[1:(N_p - |P_{g+1}|)]$; // Add the first $(N_p - |P_{g+1}|)$ elements of F_i into P_{g+1}
  end if
  $Q_{g+1} = parentsSelection(P_{g+1})$; // Get Q_{g+1} from P_{g+1} according to the binary tournament selection with //crowded-comparison operator as in NSGA-II
  $Q_{g+1} = makeNewPop(Q_{g+1})$; // Generate a new Q_{g+1} using the defined crossover and mutation operators with // constraint checking and repairing according to probabilities p_c and p_m
  $g = g + 1$;
end while
$Opt^* = \phi$; // Opt^* is the set of explicable Pareto optimal solutions
for each $X' \in F_1$ **do**
  Generate an explicable solution X^* based on X' and Eq.(7);
  $Opt^* = Opt^* \cup \{X^*\}$;
end for
Output Opt^*.

are 2 and 5 respectively, and indicates that a 2-length explicable solution can be represented as a fixed 11-length encoding. The elements with blue background in $X'(R)$ are filled with 0 and the elements with green background in $X'(P)$ are the pointers to empty set. Moreover, the element with pink background is set as 1. It is not difficult to figure out the 0-length explicable solution can also be represented when all elements in $X'(R)$ and in $X'(P)$ are respectively filled with 0 and the pointers to empty set.

Based on the above four steps, any explicable solution with the maximum length s can be represented as a $(2s+1)$-length encoding by introducing the do-nothing rule, and our fixed-length encoding does not decrease the size of the search space. What's more important, a fixed-length encoding can also reduce the complexity of evolutionary operators compared to a variable-length one, as presented by Jong (2016).

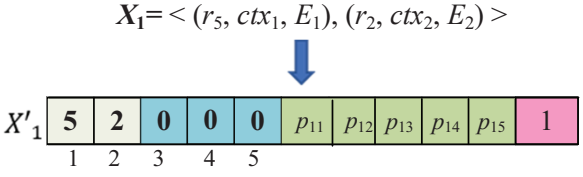


Figure 4: An example of a fixed-length encoding for an explicable solution

(3) The generation of random individuals

Based on our fixed-length encoding, an individual X' in the initial population is generated by the following three steps. Firstly, each element in $X'(P)$ is set as a pointer to the empty set, which means that the modifications in the contexts have not been computed yet. Secondly, $X'(strtPos)$ is set to 1, indicating the sequence of rules denoted by $X'(R)$ starts to execute from its first rule. Thirdly, $X'(R)$ can be initialized by using

Algorithm 2 so that it can satisfy the constraints on the maximum number of applications of each rule defined in Eq. (5).

Algorithm 2: Generation of random sequence of rule numbers

Input: The array u used to store the maximum number of applications of each rule from 0 to n
Output: The array $rulNums$ with size s used to store the sequence of rule numbers from $X'(R)$
1: Initialise the list of candidate rules $candidates$ to contain the rule numbers from 0 to n;
2: Clear the array $napplications$ with size $n+1$ used to record the number of applications of each rule from 0 to n;
3: **for** $m = 1$ to s **do**
4: k is set as a rule number randomly selected from $candidates$;
5: $rulNums[m] = k$;
6: $napplications[k] = napplications[k] + 1$;
7: **if** $(napplications[k] == u[k])$ **then**
8: Delete the rule number k from $candidates$;
9: **end if**
10: **end for**
11: Output $rulNums$.

3.3.2. *Decoding*

Decoding takes an encoding X' and the initial SA SA_0 as inputs, and outputs the corresponding objective values. During decoding X', the sequence of rules denoted by $X'(R)$ will be executed sequentially and the context for each executed rule need to be correctly built. The process of decoding is shown in Figure 5 and it includes: build the starting context, execute rules and change contexts, and output objective values (such as response time, 1-availability and cost).

(1) Build the starting context

The starting context ctx_i includes SA_{i-1} and its performance information where $i = X'(strtPos)$. The starting context can be built by two steps. Firstly, we determine SA_{i-1}. If i equals to 1, SA_{i-1} is SA_0. Otherwise, SA_{i-1} is set as SA that is obtained by sequentially applying the modifications pointed by p_1, p_2 $\cdots$, p_{i-1} to SA_0. Secondly, performance information can be got based on SA_{i-1}.

(2) Execute rules in the changing contexts

Based on the built starting context ctx_i, rule x'_i is executed and the corresponding modifications $E_{x'_i}$ can be obtained. Then, the context ctx_{i+1} for the next rule x'_{i+1} will be constructed according to ctx_i. Specifically, $E_{x'_i}$ is applied to $ctx_i[SA]$ to generate SA_i. Performance information corresponding to SA_i can be got by performance evaluation. For each of remaining rules, it is executed and the context for its next rule is built in above mentioned way.

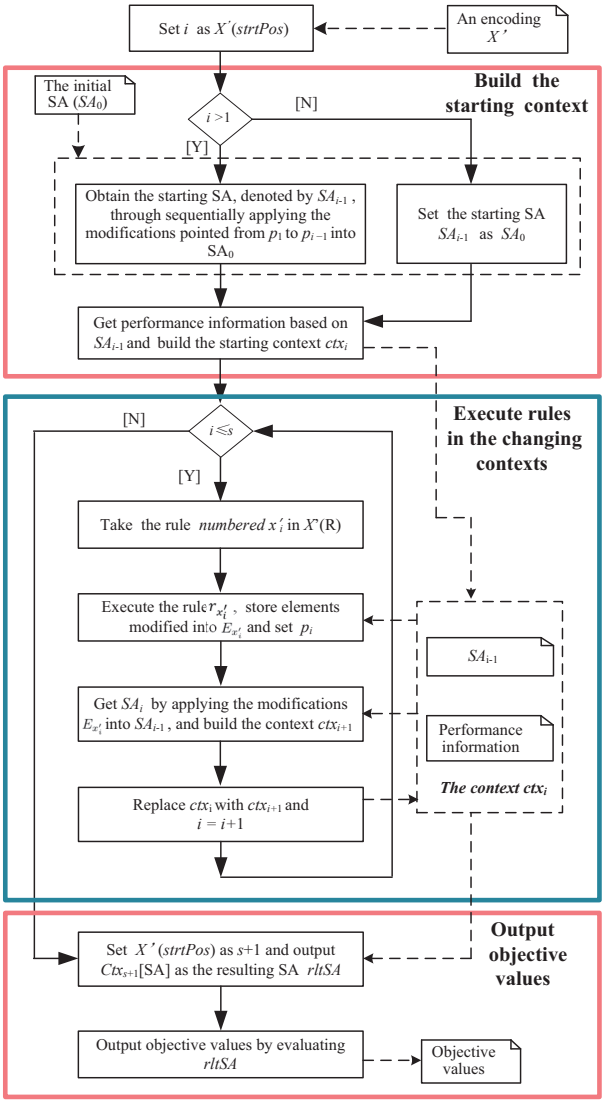


Figure 5: The process of decoding

(3) Output objective values

After all rules are executed, $X'(strtPos)$ is set to $s+1$, and the resulting SA $rltSA$ can be acquired by applying $E_{x'_s}$ into $ctx_s[SA]$. Then objective values are obtained by evaluating $rltSA$.

3.3.3. *Evolutionary operators with repair mechanism*

The evolutionary operators of MORE-EA are one-point crossover and one-point mutation with repair mechanism. Their applications, detailed descriptions

and advantages are presented, respectively.

(1) The applications of evolutionary operators with repair mechanism

The application of one-point crossover and one-point mutation with repair mechanism are presented by executing crossover or mutation operations, setting the start position of decoding, clearing incorrect modifications, constraint checking and repairing. First, the traditional one-point crossover or one-point mutation (Eiben and Smith, 2003) is performed in the $X'(R)$ of the parent individuals and intermediate individuals are generated. Second, the $X'(strtPos)$ of all intermediate individuals are set as the minimum value between the positions of crossover or mutation and the $X'(strtPos)$ of the parent individuals. Third, the modifications between the $X'(strtPos)$ position and the last position in $X'(P)$ are cleared. Fourth, each element from the $X'(strtPos)$ to the last position in the $X'(R)$ of the intermediate individuals is checked to decide whether it disobeys the constraint of the maximum number of applications of each rule defined in Eq.(5). Fifth, if the constraint is violated, these corresponding elements are set to 0. In the next paragraphs, the examples are given to illustrate the evolutionary operators of MORE-EA. Three rules r_1, r_2 and r_3 are used, and their maximum allowed number of applications are defined as $u_1 = 3$, $u_2 = 2$ and $u_3 = 3$, respectively. So the length of $X'(R)$ in each individual is 8.

(2) One-point crossover operator

The crossover operator will be applied to two individuals X'_1 and X'_2, which have already been decoded and are shown in Figure 6. The crossover point is randomly set as the 5^{th} element in the segment $X'(R)$. Two intermediate individuals X'_{11} and X'_{21} are produced by exchanging the elements from 5^{th} to 8^{th} positions in the segment $X'(R)$ of X'_1 and X'_2, and their $X'(strtPos)$ are set to 5. And p_{15}, $\cdots$,p_{18}, and p_{25}, $\cdots$, p_{28} are cleared. Then, constraint checking is done on X'_{11} and X'_{21}, as illustrated in Figure 7. The 8^{th} element in individual X'_{11} and the 5^{th} element in individual X'_{21} disobey the constraints defined by Eq.(5), because $h_3(X'_{11}) = 4 > u_3$ and $h_2(X'_{21}) = 3 > u_2$. Therefore, these two elements are set to 0. As a result, two offspring individuals X'_3 and X'_4 are generated as shown in Figure 8.

(3) One-point mutation operator

Suppose one-point mutation is applied to an individual X'_4 obtained through the above crossover operator. The mutation point is selected randomly as 2 which means mutation will occur at the 2^{nd} element in the segment $X'(R)$. The rule number 2 is chosen uniformly at random to replace rule number 1 at the mutation point. Then the intermediate individual is obtained and its $X'(strtPos)$ is set to 2. And the modified elements p_{23}, $\cdots$, p_{28} are cleared. Furthermore, constraint checking is performed from the 2^{nd} element to the 8^{th} element in the $X'(R)$ of X'_{41}. The 3^{rd} element disobeys the constrains defined by Eq.(5) because $h_2(X'_{41}) = 3 > u_2$, as illustrated in Figure 9. Finally, the 3^{rd} element is set to 0. Then the offspring individual named X'_5 is formed as shown in Figure 10.

(4) The advantages of our evolutionary operators with repair mechanism

Based on our proposed repair mechanism, the invalid encodings generated by crossover and mutation can be transformed into valid ones by use of the do-nothing rule. It brings two advantages: firstly, the efficiency of MORE-EA can be improved due to the repair mechanism. This is a consensus in the field of evolutionary computation (Salcedo-Sanz, 2009): the EAs with repair mechanism perform better than other kind of EAs in handling optimisation problem with constraints. Secondly, our repair mechanism can potentially improve explanations for optimisation results. This will be elaborated as follows. It is possible that some encodings derived from evolutionary operator with repair mechanism are in the set of Pareto optimal encodings. When they are transformed into explicable solutions by the postprocessing step (See lines 23-26 in Algorithm 1), the rule applications associated with the do-nothing rule will be removed and explanations for results can be improved based on the metrics defined in Table (5). More details can be found in the definition of explicable solution in Section 3.2.2.

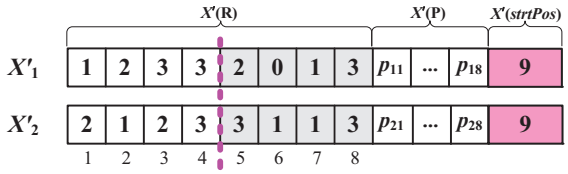


Figure 6: Two parent individuals before crossover at the 5^{th} position

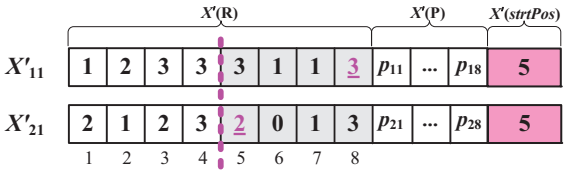


Figure 7: Two intermediate crossover individuals and their constraint checking

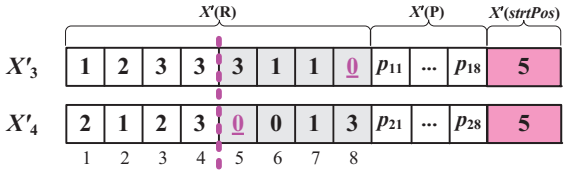


Figure 8: Two offspring individuals generated through crossover and repair

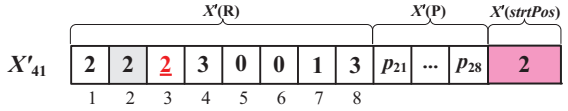


Figure 9: Intermediate mutated individual generated and their constraint checking

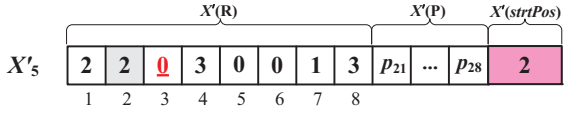


Figure 10: Offspring individual generated through mutation and repair

3.3.4. Computational Complexity of MORE-EA

MORE-EA is a multi-objective evolutionary algorithm (MOEA). The main difference between MORE-EA and traditional MOEA is solving objective value which is acquired by decoding. The process of decoding of MORE-EA in Figure 5 includes 3 stages: build the starting context, execute rules in the changing contexts, and output objective values (such as response time, 1-availability and cost). Among them, obtaining performance information in the stage 2 is the most time-consuming. Specifically, traditional MOEA only needs to obtain performance information once when solving objective value. However, MORE-EA needs to obtain performance information many times when solving objective value. The number of acquirement of performance information in MORE-EA relates to $X'(strtPos)$ in $X'(R)$ shown in Figure 5. Suppose that $X'(strtPos)$ in $X'(R)$ of offspring individual generated through crossover or mutation has equal probability in each position. Then, the average number of obtaining performance information is equal to $\frac{1}{s}\sum_{i=1}^{s}(s-i+1) = \frac{s+1}{2}$, where s is the size of $X'(R)$. So, the computational complexity of solving objective value in MORE-EA is $s/2$ times that of traditional MOEA.

4. Experimental design

This section demonstrates the design of our experimental study. Specifically, Section 4.1 presents the research questions we set out to answer and the assessment methods. Section 4.2 introduces the statistical test methods used to support our analysis. Section 4.3 gives information about the six problem instances that we have used. Section 4.4 presents the experimental setup. In order to explain the comparison results conveniently, the explicable solutions and the set of explicable Pareto optimal solutions obtained by MORE in Section 4 and Section 5 are called the solutions and the set of Pareto optimal solutions, respectively.

4.1. Research questions and assessment methods

MORE is a rule-based method. And it adopts metaheuristic algorithm to search the best solutions. Therefore, MORE has the features of both rule-based and metaheuristica-based methods. In order to validate the effectiveness of MORE, a representative metaheuristic-based method (PCM) in Section 2.3.2 and a typical rule-based method (PB) in Section 2.3.3 are selected as compared methods for experimental study. The two comparisons, PB *vs* MORE and PCM *vs* MORE, need to be separately done. This is because there are several differences in the two comparisons, including

the adopted rules, the optimisation objectives, the parameter settings, the DoFs and the problem instances. Therefore, we set four research questions for quality of solutions and explanation of the results, and defined the corresponding assessment criterion to answer these questions as follows.

4.1.1. RQ1: Comparison between PCM and MORE on the quality of solutions

How effective is MORE compared to PCM in terms of response time, cost and availability? For MORE to be adopted, its exploration of the space defined by randomised search rules should lead to better solutions than PCM method.

To provide the quantitative assessment for this question, we employ three quality indicators to evaluate the performance of the concerned algorithms, namely Contribution (I_C), Generational Distance (I_{GD}) and Hypervolume (I_{HV}). To compute these indicators, we normalise objective values to avoid unwanted scaling effects (Durillo and Nebro, 2011) .

The contribution CONT defined in (Meunier et al., 2000) is the contribution that quantifies the domination between two sets of non dominated solutions. It is represented as I_C. Let A_1 and A_2 be two compared algorithms. The contribution $CONT(\frac{A_1}{A_2})$ of algorithm A_1 relatively to A_2 is roughly the ratio of non dominated solutions produced by A_1. Let S be the set of solutions in $A_1 \cap A_2$. Let W_1 be the set of solutions by A_1 that dominate at least one solution of A_2. Similarly, let L_1 be the set of solutions by A_1 that are dominated by at least one solution of A_2. The set of solutions in A_1 that are not comparable to solutions in A_2 is $N_1 = A_1 \setminus \{S \cup W_1 \cup L_1\}$. The set of solutions in A_2 that are not comparable to solutions in A_1 is $N_2 = A_2 \setminus \{S \cup W_2 \cup L_2\}$ where the definitions of W_1 and L_1 are similar to W_2's and L_2's. The contribution I_C is defined by Eq.(10). The higher I_C, the better A_1's solution set is in comparison to A_2's.

$$I_C = CONT(\frac{A_1}{A_2}) = \frac{|S|/2 + |W_1| + |N_1|}{|S| + |W_1| + |N_1| + |W_2| + |N_2|} \quad (10)$$

The I_{GD} quality indicator (Zitzler et al., 2003) computes the average distance in the objective space between the set S and the reference set RS. S is the set of non-dominated points obtained by the measured algorithm. RS is the set of non-dominated points obtained by the union of all measured algorithms (Fonseca et al., 2005). The average distance between S and RS in a m objective space is computed as the average m-dimensional Euclidean distance between each point in S and its nearest neighbouring point in RS. To keep consistent with other two quality indicators that the larger I_{GD} value means better, we modify the original I_{GD}. The modified I_{GD} is defined in Eq.(11), where $d(x, y)$ is the Euclidean distance between the points x and y.

$$I_{GD}(S) = exp(-\frac{\Sigma_{y \in RS} min_{x \in S}(d(x, y))}{|RS|}) \quad (11)$$

The I_{HV} quality indicator (Zitzler and Thiele, 1999) calculates the volume (in the objective space) covered by members of a non-dominated set of solutions from an algorithm of interest. Zitzler et al. (2003) demonstrates that this hypervolume measure is also strictly 'Pareto compliant', whose nice theoretical qualities make it a rather fair indicator. Let S be the set of final nondominated points obtained in the objective space by an algorithm, and $r = (r_1, r_2, \cdots r_m)^T$ be a reference point in the objective space which is dominated by any point in the set S. Then the hypervolume indicator value of S with regard to r is the volume of the region dominated by S and bounded by r, and can be described as Eq.(12). Given a reference point r, the larger I_{HV} value means better quality. In our experiments, r is selected from all the solution sets obtained by the union of compared algorithms for a certain problem instance. $f = [f_1, \cdots, f_m]^T$ is used to store m objective values obtained.

$$I_{HV}(S, r) = volume\left(\bigcup_{f \in S} [f_1, r_1] \times \cdots [f_m, r_m]\right) \quad (12)$$

4.1.2. RQ2: Comparison between PB and MORE on the quality of solutions

How does MORE perform in comparison to PB method (including PB_{cost} and PB_{time}) in terms of response time and cost?

PB_{cost} and PB_{time}, introduced in Section 2.3.3, are two subcategories of PB methods that adopt cost-first and RT-first (response time first) search strategies to obtain the single best solution, respectively.

As explained in Section 2, availability is not considered by the PB methods (both PB_{cost} and PB_{time}) and the two problem instances from PB's experiments (Xu, 2012) have no specification for availability. Therefore, it is impossible for us to obtain the objective value of availability for PB methods. And RQ2 only evaluates MORE's solutions in comparison to PB methods in terms of response time and cost.

$$\begin{aligned}\eta_{cost} &= \{X^* \mid X^* \in Opt^* \wedge time(X^*) \leq time(s_{cost}) \wedge \\ &\qquad cost(X^*) < cost(s_{cost})\} \\ \eta_{time} &= \{X^* \mid X^* \in Opt^* \wedge time(X^*) < time(s_{time}) \wedge \\ &\qquad cost(X^*) \leq cost(s_{time})\} \end{aligned} \quad (13)$$

$$DR_{cost} = \frac{|\eta_{cost}|}{|Opt^*|}$$

$$DR_{time} = \frac{|\eta_{time}|}{|Opt^*|}$$

To answer RQ2 in a quantitative manner, the dominance ratios DR_{cost} and DR_{time} are proposed as two measurements and formally defined as Eq.(13). DR_{cost} and DR_{time} are the percentages of solutions which are better than PB_{cost}'s and PB_{time}'s, respectively, in the set of Pareto optimal solutions obtained by MORE.

In Eq.(13), s_{cost} and s_{time} are the solutions obtained by PB_{cost} and PB_{time}, respectively. Opt^* is the set of Pareto optimal solutions acquired by MORE. The functions $time$ and $cost$ return the objective values of response time and cost of a solution, respectively.

If DR_{cost} and DR_{time} are both larger than 0, it means that MORE finds at least one solution that is better than PB_{cost}'s and at least one solution that is better than PB_{time}'s. Then we denote the percentages of maximal, average and minimum improvement of response time and cost in comparison with PB_{cost} and PB_{time} as $time_{max}\%$, $time_{avg}\%$, $time_{min}\%$, $cost_{max}\%$, $cost_{avg}\%$, $cost_{min}\%$, respectively. These metrics will be used to evaluate how well MORE does in comparison with PB_{cost} and PB_{time}.

4.1.3. RQ3: Comparison between PCM and MORE on the explanations for results

How does MORE perform compared to PCM in terms of the explanations for results? While outperforming PCM may be a valuable technical result, MORE should also enable a better explanation on how to obtain a resulting SA from the initial SA.

Considering that PCM only has the trivial explicability, the metric $AvgNumMdf$ shown in Table 5 of Section 3.2.1 is used to compare the explanations for results between MORE and PCM. This metric is defined as the average number of modifications to the initial SA in the set of optimal solutions.

However, it is worth noting that, MORE can provide more useful explanations to software architects by informing them of the sequence of randomised search rules applied to the initial SA and the modified elements after the application of each rule. This sequence and their corresponding modifications provide good explanatory power because each randomised rule is derived from anti-patterns or design tactics provided by the performance optimisation practices.

4.1.4. RQ4: Comparison between PB and MORE on the explanations for results

How does MORE perform compared to PB_{cost} and PB_{time} in terms of the explanations for results? Software architects incline to adopt MORE, if MORE can acquire a few solutions that not only have better response time and lower cost, but also better explicability than PB_{cost} and PB_{time}.

For quantitatively comparing the explicability given by MORE and PB methods, the metrics of $avgNumRul_+$ and $avgNumMdf_+$ shown in Table 5 of Section 3.2.1 are used. The $avgNumRul_+$ and $avgNumMdf_+$ are defined as the average number of rule applications in the set of optimal solutions, and the average number of modifications in rule applications in

the set of optimal solutions, respectively.

PB_{cost} and PB_{time} can only output a single best solution in their sets of optimal solutions. In comparisons of MORE *vs* PB_{cost} and MORE *vs* PB_{time} on $avgNumRul_+$ and $avgNumMdf_+$, η_{cost} and η_{time} defined in Eq.(13) are regarded as the sets of optimal solutions for MORE, respectively.

4.2. Statistical tests

In this paper, nonparametric statistical tests are used to analyze the results, as recommended by Arcuri and Briand (2011). To compare groups, the nonparametric Wilcoxon rank-sum test (Arcuri and Briand, 2011) is used to check for statistical significance. Wilcoxon test is a safe test to apply (even for normally distributed data), since it raises the bar for significance, by making no assumptions about underlying data distributions. We set the level of significance α at 0.05.

Furthermore, we use the Vargha-Delaney $\hat{A}_{12}$ metric for effect size, as recommended by Grissom and Kim (2005). Vargha-Delaney $\hat{A}_{12}$ also makes few assumptions and is highly intuitive. $\hat{A}_{12}(A_1, A_2)$ is simply the probability that algorithm A_1 will outperform algorithm A_2 in a head-to-head comparison.

4.3. Problem instances

To obtain convincing experimental results, we should guarantee the diversity of problem instances. Therefore, six problem instances with different scale and categories have been selected in our experiments: STE (Software Design and Quality Group, Karlsruhe Institute of Technology), MS (Brosch et al., 2011), BRS (Koziolek et al., 2013), PCS (Koziolek et al., 2011b), WebApp (Xu, 2012) and IES (Xu, 2012). The first four problem instances are described in Palladio component model and annotated with specification of performance, availability and cost. Therefore, they can be used in comparison between MORE and PCM. Instead, the last two problem instances are specified by LQN model and have no annotation for availability. So, we use the two problem instances when comparing MORE with PB.

All problem instances are real world applications or systems, except for STE, which is an illustrative one. At the same time, these problem instances cover various categories, such as enterprise information system , industrial processes management and images processing and so on. Table 7 shows the complexity and scale of the six problem instances. It can be seen that six problem instances have different scale, as shown in the last column that indicates the lower bound of size of search space for each problem instance. Each lower bound was estimated by the number of instances of DoFs in the initial SA of each problem and the range of value of each instance.

Table 7
Comparison of the complexity and scale of the six problem instances

Compared method	Problem instance	$N.S.$	$N.C.$	Category	$\|\Omega\|$
PCM	STE	3	3	IMS	$> 10^5$
	MS	3	4	Security	$> 10^7$
	BRS	4	9	EIS	$> 10^9$
	PCS	5	11	IPM	$> 10^{12}$
PB	WebApp	5	6	Web application	$> 10^7$
	IES	4	10	Images processing	$> 10^9$

$N.S.$: The number of Servers
$N.C.$: The number of Components
$|\Omega|$: The size of search space
IMS: Information management system
IPM: Industrial processes management
EIS: Enterprise information system

4.3.1. STE

A business trip management system is used as Simple Tactics Example (STE) (Software Design and Quality Group, Karlsruhe Institute of Technology) when presenting the optimisation tool of PCM method. In this problem instance, the user's business trip can be efficiently arranged by three subsystems of booking, payment and business management. Resource demands were defined as an illustrative example. Figure 11 gives the initial deployment view of STE.

4.3.2. MS

MS (Brosch et al., 2011) is a plain Java web application for storing and retrieving audio or video files using a MySQL database. The model reflects a use

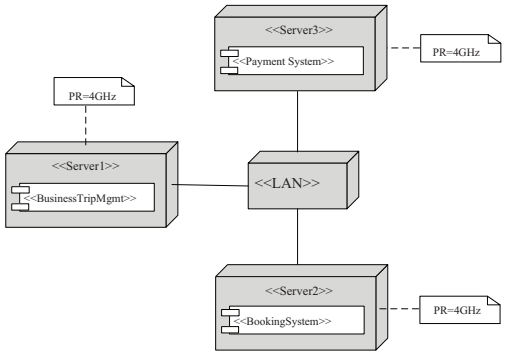


Figure 11: The initial deployment view of STE

case where a digital watermark is added to download files for copy protection. The model contains a hard disk resource accessed when retrieving files. Resource demands for the Media Store have been measured using manual instrumentation of the Java implementation. Figure 12 shows the initial deployment view of MS.

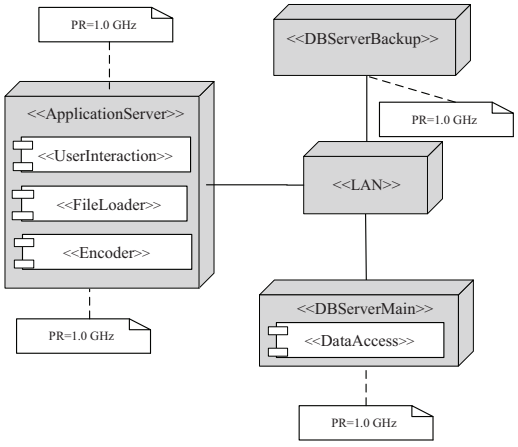


Figure 12: The initial deployment view of MS

4.3.3. BRS

Business Reporting System (BRS) (Koziolek et al., 2013) lets users retrieve reports and statistical data about running business processes from a data base. It represents a typical enterprise system. The analyzed configuration comprises two usage scenarios, nine components, and four servers. Users can retrieve live business data from the system and run statistical analyses. Resource demands of the BRS are based on estimations. Figure 20 (a) shows the initial deployment view of BRS.

4.3.4. PCS

Process control system (PCS) (Koziolek et al., 2011b) is a distributed system to manage industrial processes, such as power generation, oil refinement, or pulp and paper processing. The model focuses on the server-side part of the system, which is implemented in C++ using Microsoft technologies. It features four usage scenarios, for example for transferring sensor data or managing alarm events. Resource demands were determined using Windows Performance Monitor. Figure 13 presents the initial deployment view of PCS.

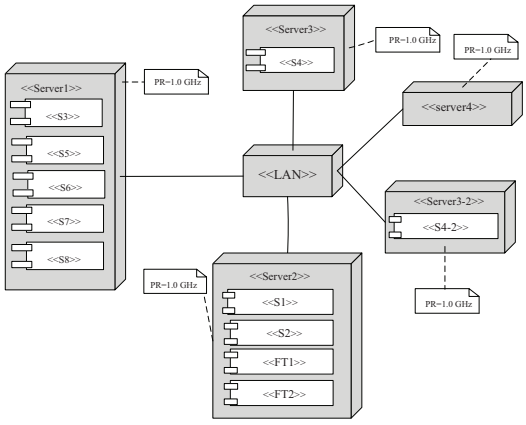


Figure 13: The initial deployment view of PCS

4.3.5. WebApp

In WebApp (Xu, 2012), users access the pages deployed on Web Server (WS) through their browser. When the WS loads and deals with these pages, it will call business services deployed on application servers App1 and App2. These business services will call data management services deployed on database servers DB1 and DB2. Resource demands were determined by the architect. Its initial LQN model is shown in Figure 23 (a).

4.3.6. IES

IES (Xu, 2012) processes images bearing credential information such as scans of identification cards, signatures, or legal documents. It converts images to a common format, encrypts them and places them on a public server for a partner to pick up. An operator

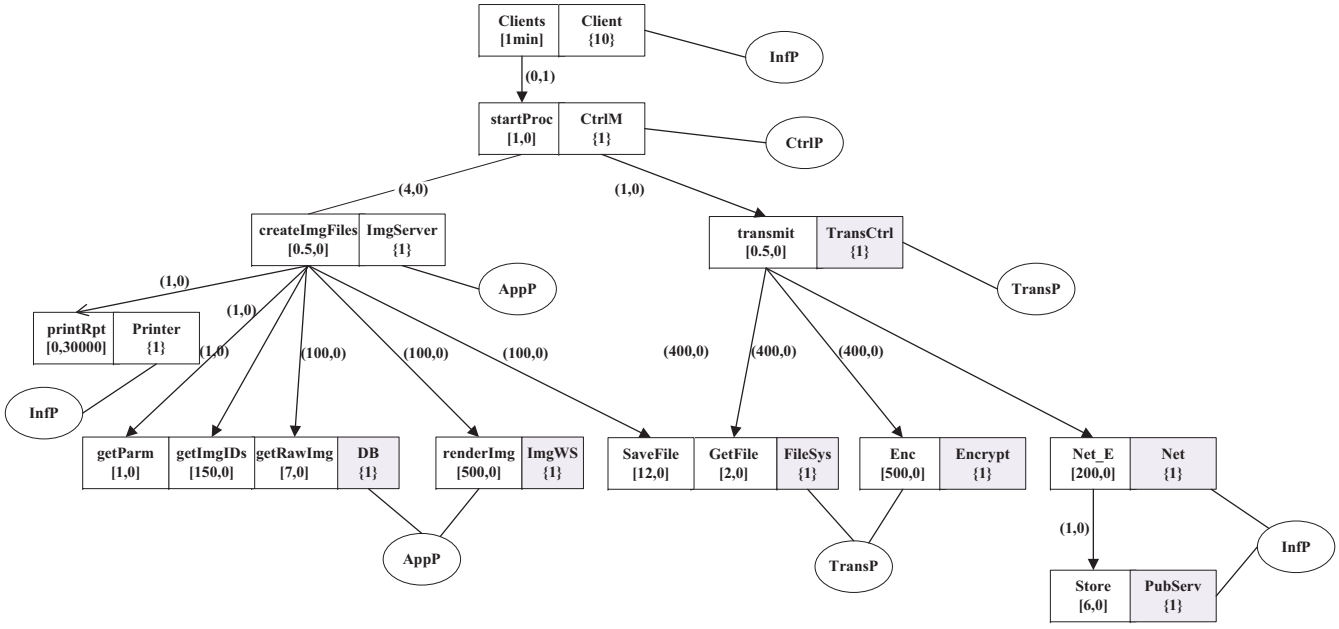


Figure 14: The initial LQN model of IES (Xu, 2012)

should be able to process a batch of images every 15 minutes, which we will call the cycle time c ($c < 900$ seconds). This gives a throughput requirement for ten sites of $f = 10/c > 10/900/$ s. Resource demands were defined by the architect. Figure 14 shows the initial LQN model of IES.

4.4. Experimental setup

To compare MORE with PCM and PB, we need to do three works. Firstly, we define the randomised search rules, and this has been done in Section 3.1.3 by selection and addition of rules for MORE. Secondly, we set the maximum allowed number of applications of each selected rule based on the heuristics described in the definition of explicable solution in Section 3.2.2. Table 8 shows the selected and added rules and their maximum allowed number of applications in the problem instances of STE, MS, BRS and PCS. Instead, Table 9 shows the selected rules and their maximum allowed number of applications in the problem instances of WebApp and IES. Here, we take BRS as an example to demonstrate how to set the maximum allowed number of applications of selected rule. As shown in Figure 20 (a), there are 4 severs (i.e., $server1, server2, server3$ and $server4$) and 9 components in the initial SA of BRS. Rule 1 is related to allocation of components (AC) and is selected shown in Table 8. The range of each instance of AC is $\{server1, server2, server3, server4\}$. Each component has the chance to be reallocated to these 4 severs according to the action of Rule 1. Therefore, the number of each instance of AC is considered as the value of the maximum allowed number of applications of Rule 1 in BRS. The cell in the first row and the fourth column in Table 8 is filled with 4. Similarly, the maximum allowed numbers of applications of other selected Rules in Table 8 and Table 9 can also be filled. Thirdly, we set the parameters of each compared method. PCM's and PB's parameter settings are same as the ones used in their corresponding papers (Koziolek et al., 2013; Xu, 2012). The parameters of MORE-EA are set to the same values used in the PCM paper (Koziolek et al., 2013) and are shown in Table 10.

For each problem instance, MORE-EA and PCM are independently run 30 times. While PB_{cost} and PB_{time} are deterministic and so only one run is necessary.

5. Experimental results and analysis

This section presents the experimental results and analysis done to answer the four research questions pro-

Table 8
The selected and added rules and their maximum allowed number of applications in the problem instances of BRS,MS, PCS and STE

Rule	STE	MS	BRS	PCS
Rule 1	3	6	4	5
Rule 3	9	9	10	10
Rule 5	1	2	2	3
Rule 11	1	3	2	1
Rule 18	2	3	2	5
Rule 19	2	3	2	2

Table 9
The selected rules and their maximum allowed number of applications in the problem instances of WebApp and IES

Rule	WebApp	IES
Rule 1	3	3
Rule 4	3	3
Rule 6	12	12
Rule 7	1	1
Rule 9	3	3
Rule 11	6	3
Rule 13	6	3

Table 10
The parameter settings of MORE-EA and PCM

Parameter	MORE-EA	PCM(NSGA-II)
Population size	30	30
Crossover rate	0.8	0.8
Mutation rate	0.6	0.6
Max generations	200	200

posed in Section 4.1.

5.1. Results for RQ1 (Comparison between PCM and MORE on the quality of solutions)

Table 11 shows that MORE achieved superior mean values for all three quality indicators (I_C, I_{GD} and I_{HV}) compared to PCM, on all four problem instances (STE, MS, BRS and PCS). The Wilcoxon rank−sum tests reported in Table 12 confirm that all these differences are significant (p-value <<0.05), which can also be seen from Figure 15. At the same time, the values of the Vargha-Delaney $\hat{A}_{12}$ effect size in Table 13 show that to a very high extent, MORE outperforms PCM in all 12 experiments under the three indicators and the four problem instances.

Table 11
Mean values of the quality indicators of I_C, I_{GD} and I_{HV} for PCM and MORE on the problem instances of STE, MS, BRS and PCS. MORE is significantly better than PCM at the level of significance of 0.05 on all these values, according to the Wilcoxon Rank−sum tests shown in Table 12.

problem	I_C		I_{GD}		I_{HV}	
instance	**PCM**	**MORE**	**PCM**	**MORE**	**PCM**	**MORE**
STE	0.1070	0.8930	0.9762	0.9960	0.8429	0.8615
MS	0.1526	0.8474	0.9634	0.9872	0.8224	0.8536
BRS	0.2089	0.7911	0.9685	0.9880	0.8374	0.8656
PCS	0.1520	0.8480	0.8474	0.9703	0.4912	0.6743

Table 12
The p-values of the Wilcoxon Rank−Sum tests to compare MORE against PCM in terms of quality indicators contribution I_C, generational distance I_{GD} and hypervolume I_{HV} on the problem instances of STE, MS, BRS and PCS. All p-values show statistically the significant difference at the level of 0.05.

problem instance	I_C	I_{GD}	I_{HV}
STE	3.0199E-11	0.0003E-07	0.0024E-07
MS	3.0199E-11	0.0098E-07	0.0003E-07
BRS	14.6431E-11	33.2415E-07	1174.7192
PCS	2.9822E-11	0.0003E-07	0.0003E-07

Table 13
Effect size $\hat{A}_{12}$ of the quality indicators of I_C, I_{GD} and I_{HV} for PCM and MORE on the four problem instances of STE, MS, BRS and PCS

problem instance	I_C	I_{GD}	I_{HV}
STE	1.0000	1.0000	0.9767
MS	1.0000	0.9933	1.0000
BRS	0.9822	0.8500	0.7900
PCS	1.0000	1.0000	1.0000

In more detail, we observe that for all four problem instances, the values of the effect size of I_C are higher than those of I_{GD} and I_{HV}, which are reflected in their relative positions of boxes shown in Figure 15(a). Compared to other three problem instances, the

lowest value of the effect size was obtained for BRS as shown in Figure 15(b). Whilst, the outliers, which are achieved by MORE in BRS and STE problem instances and presented in Figure 15(c), contribute to the lower values of effect size of I_{HV}. The consistent results can be intuitively and qualitatively observed from the reference Pareto surfaces obtained by PCM and MORE for four problem instances and shown in Figure 16.

5.2. Results for RQ2 (Comparison between PB and MORE on the quality of solutions)

The single best response time and cost obtained by PB_{cost} and PB_{time} for WebApp and IES are shown in Table 14, respectively.

Table 14
The best response time and cost obtained by PB_{cost} and PB_{time} in WebApp and IES problem instances

	WebApp		IES	
	response time	cost	response time	cost
PB_{cost}	35.5	17	199902	57
PB_{time}	29.88	25	199783	60

Wilcoxon Rank−Sum tests were performed to determine whether $DR_{cost} > 0$ and $DR_{time} > 0$. The results are shown in Table 15, and confirm that MORE can achieve at least a few better solutions in the set of Pareto optimal solutions, compared to PB_{cost} and PB_{time} for both WebApp and IES. The best, average and worst percentage of MORE's solutions which are better than PB_{cost}'s and PB_{time}'s in the set of Pareto optimal solutions for WebApp and IES are presented in Table 16. From Table 16, we observe that MORE always obtained no less than 20% of solutions which are better than PB_{cost}'s and no less than 15% of solutions which are better than PB_{time}'s in the set of Pareto optimal solutions. On average, for both WebApp and IES, MORE obtained more than 44% solutions in the set of Pareto optimal solutions which are better than PB_{cost} and PB_{time}.

For the solutions obtained by MORE and are better than PB_{cost}'s and PB_{time}'s, their objective values of response time and cost compared with PB_{cost}'s and

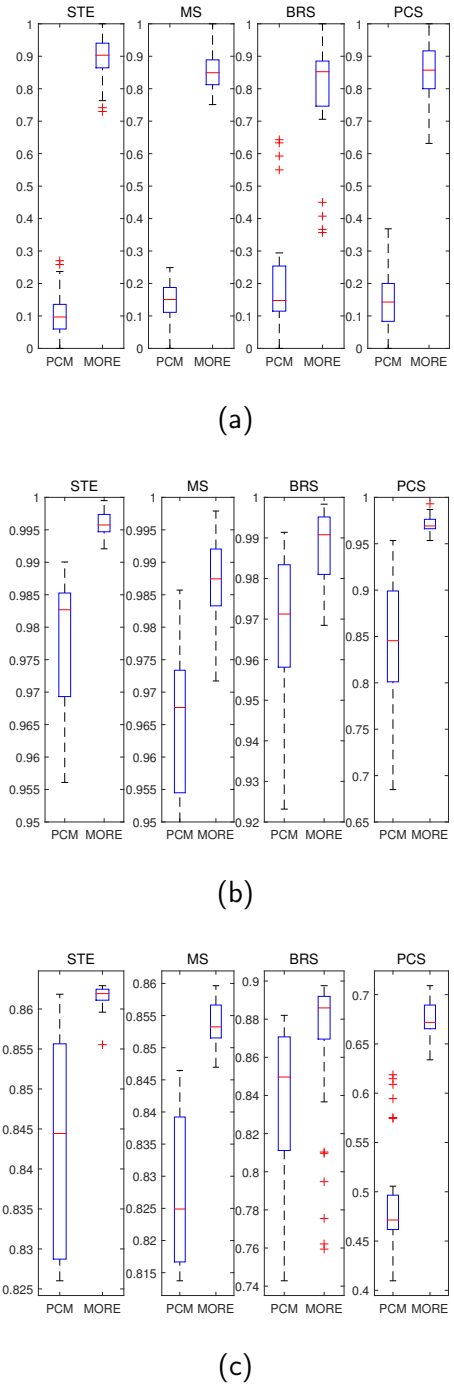


Figure 15: Boxplots using the quality indicators I_C (a), I_{GD} (b) and I_{HV} (c) applied to PCM (left) and MORE (right) for the four problem instances STE, MS, BRS and PCS, respectively.

PB_{time} 's are presented in Figures 17 and 18. Furthermore, Table 17 shows the percentages of the maximum, average and minimum improvement of response time and cost by comparing MORE's solutions with PB_{cost}'s and PB_{time}'s. It can be seen that the response

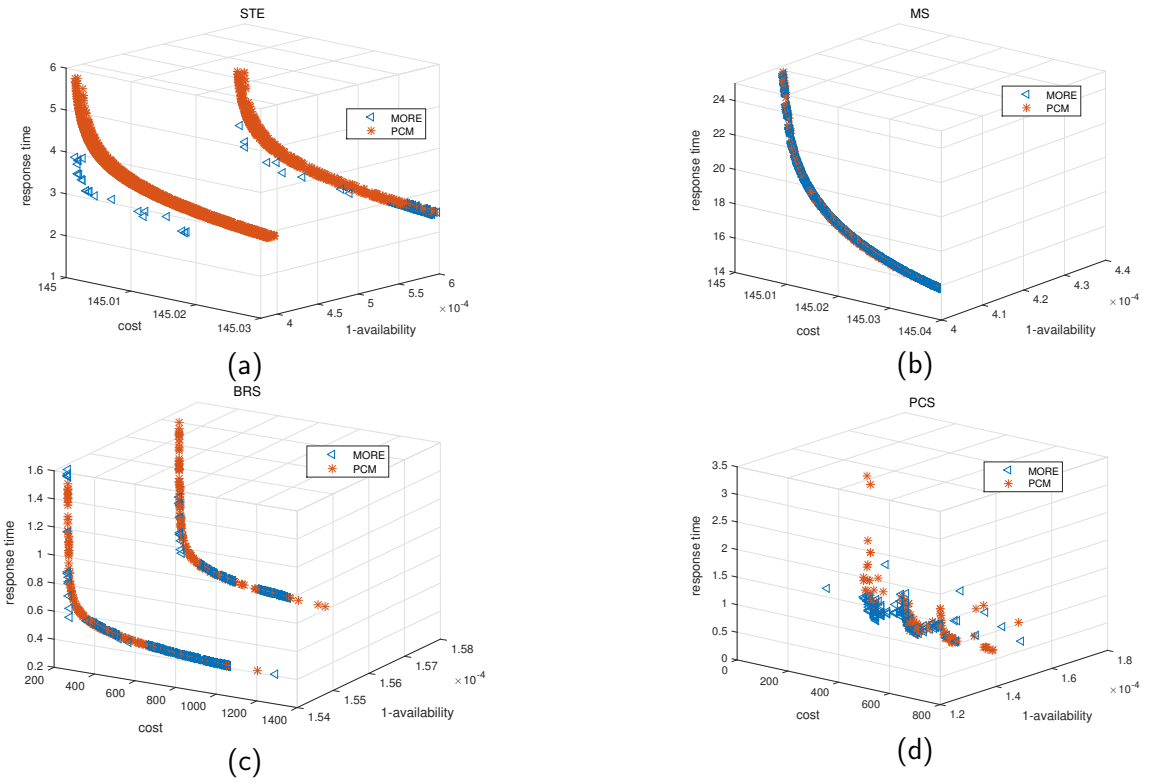


Figure 16: Pareto surfaces obtained by PCM (depicted by the stars) and MORE (depicted by the triangles) for four problem instances of STE (a), MS (b), BRS (c) and PCS (d)

Table 15
The p-values of the statistical experiments of $DR_{cost} > 0$ and $DR_{time} > 0$ on the problem instances of WebApp and IES for comparing MORE against PB_{cost} and PB_{time} at the level of significance of 0.05

experiments	p-value	
	WebApp	IES
$DR_{cost} > 0$	1.2118E-12	1.2118E-12
$DR_{time} > 0$	1.2118E-12	1.2118E-12

Table 16
The best, average and worst values of DR_{cost} and DR_{time} (the percentage of solutions which are better than PB_{cost} and PB_{time} in the set of Pareto optimal solutions obtained by MORE) for WebApp and IES problem instances, considering 30 runs

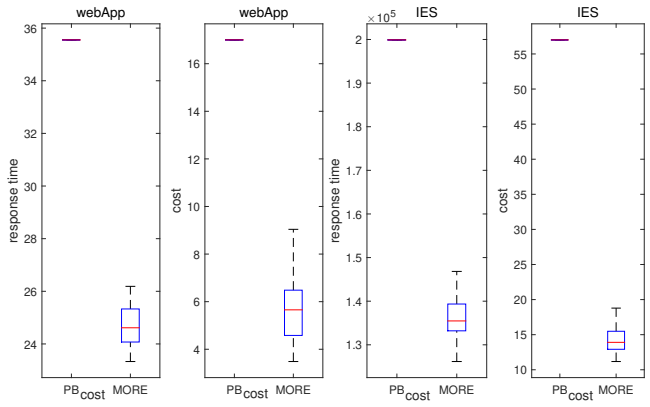
	WebApp			IES		
	best	average	worst	best	average	worst
DR_{cost}	94%	61%	20%	65%	44%	33%
DR_{time}	100%	59%	15%	65%	44%	33%

time and cost can be improved by no less than 20% and 67% in the average case, respectively.

5.3. Results for RQ3 (Comparison between PCM and MORE on the explanations for results)

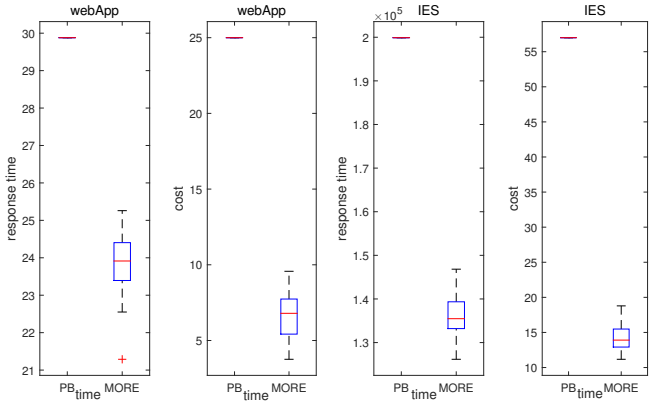
The mean $AvgNumMdf$ across 30 runs are shown in Table 18. The results of Wilcoxon Rank−Sum test from Table 19 show that, for the four problem instances of STE, MS, BRS and PCS, the values of $AvgNumMdf$ for MORE were significantly better than those of PCM. Figure 19 further confirms that MORE outperforms PCM in terms of $AvgNumMdf$ with the highest effect size of $\hat{A}_{12} = 1$ in all problem instances. Furthermore, Table 18 shows that the decreasing $AvgNumMdf$ ratio of MORE to PCM, $\frac{AvgNumMdf_{PCM} - AvgNumMdf_{MORE}}{AvgNumMdf_{PCM}}$, varies from 17.5% in BRS to 52.6% in MS.

Besides using the number of elements modified from the initial SA as a kind of trivial explicability, the



(a) (b)

Figure 17: Boxplots using the response time and cost to evaluate the quality of solutions of PB_{cost} (left) and MORE (right) in the problem instances of WebApp (a) and IES (b)



(a) (b)

Figure 18: Boxplots using the response time and cost to evaluate the quality of solutions of PB_{time} (left) and MORE (right) in the problem instances of WebApp (a) and IES (b)

full explicability derived from predefined randomised search rules in MORE can be further presented through an example. In this example, the two solutions with the best response time are selected from the Pareto optimal sets obtained by MORE and PCM in BRS, respectively. Based on the two solutions, Figure 20 shows the resulting deployment views of BRS obtained by PCM and MORE. In Figure 20, the modified elements with respect to the initial SA are marked in red. And it can be seen that MORE modified fewer elements than PCM. Moreover, unlike PCM, MORE can also explain how

Table 17
Mean values of $time_{max}\%$, $time_{avg}\%$, $time_{min}\%$, $cost_{max}\%$, $cost_{avg}\%$ and $cost_{min}\%$ (the percentage of the maximum, average and minimum improvement of response time and cost by comparing MORE with PB_{cost} and PB_{time}, respectively) for WebApp and IES, considering 30 runs

the percentage of improvement	WebApp		IES	
	PB_{cost}	PB_{time}	PB_{cost}	PB_{time}
$time_{max}\%$	43%	33%	41%	41%
$time_{avg}\%$	30%	20%	32%	32%
$time_{min}\%$	12%	5%	7%	7%
$cost_{max}\%$	87%	90%	86%	87%
$cost_{avg}\%$	67%	73%	75%	76%
$cost_{min}\%$	27%	37%	48%	51%

to obtain the resulting SA from the initial SA by applying the predefined rules. As shown in Table 20, through the 12 steps, the resulting SA of BRS can be obtained, explained and even replayed. Each step gives the cause for objective improvement, defined by the condition part of rule, and the modified architectural elements and their values denoted by the action part of rule. From Table 20, we observe the predefined rules of Rule 1, Rule 3, Rule 5, Rule 11, Rule 18, and Rule 19 are all applied and the number of applications of each rule is 4, 3, 1, 2, 1 and 1, respectively. Rule1 (increasing processing rate) is applied more times because it has relatively larger search space than other rules.

Table 18
The mean values of $AvgNumMdf$ by PCM and MORE in the four problem instances of STE, MS, BRS and PCS. MORE is significantly better than PCM at the level of significance of 0.05 in all problem instances, according to the Wilcoxon Rank–Sum tests shown in Table 19.

	STE	MS	BRS	PCS
PCM	5.82	10.58	12.36	14.44
MORE	4.33	5.02	10.20	10.64
Decreasing the ratio of MORE to PCM	25.6%	52.6%	17.5%	26.3%

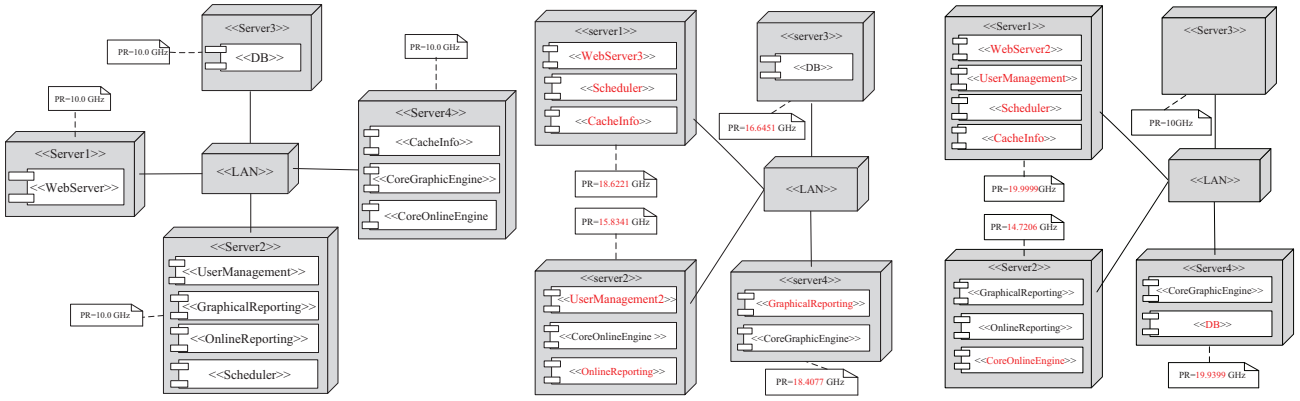


Figure 20: The resulting deployment views of BRS obtained by PCM(b) and MORE(c) against the initial deployment view (a) with respect to the solutions with the best response time in the set of Pareto optimal solutions obtained by PCM and MORE, respectively

Table 19
The p-values of the Wilcoxon rank−sum tests to compare MORE against PCM in terms of $AvgNumMdf$ on the problem instances of STE, MS, BRS and PCS. All p-values show statistically significant difference at the level of 0.05.

problem instance	p-value
STE	3.0161E-11
MS	3.0199E-11
BRS	3.0123E-11
PCS	3.0029E-11

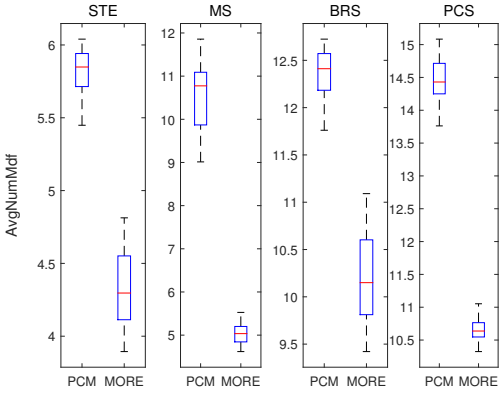


Figure 19: Boxplots of the average number of elements modified by PCM (left) and MORE (right) for the four problem instances STE, MS, BRS and PCS, respectively.

5.4. Results for RQ4 (Comparison between PB and MORE on the explanations for results)

The mean $avgNumRul_+$ and $avgNumMdf_+$ across 30 runs for MORE, PB_{cost} and PB_{time} are shown in Tables 21 and 22, respectively. Table 23 presents the results of Wilcoxon rank-sum tests to compare MORE against PB_{cost} and PB_{time}. They show that MORE is significantly better than PB_{cost} and PB_{time} with respect to the metrics of $avgNumRul_+$ and $avgNumMdf_+$.

From Table 21, we can see that $avgNumRul_+$ and $avgNumMdf_+$ are improved by at least 33% and 53% by MORE with respect to PB_{cost}, respectively. From Table 22, we observe that MORE reduces $avgNumRul_+$ and $avgNumMdf_+$ by at least 33% and 35% against PB_{time}. Figures 21 and 22 further illustrate the significantly better $avgNumRul_+$ and $avgNumMdf_+$ obtained by MORE in comparison with PB_{cost}'s and PB_{time}'s.

Table 20
An example for BRS on how to obtain the resulting SA from the initial SA by applying the predefined rules, based on the solution with the best response time obtained by MORE

Step	Rule	Explanation
1	Rule 1	Component UserManagement is redeployed on server1 from server2 because of the highest utilisation of server2
2	Rule 11	Component WebServer is replaced by its alternative component WebServer3 because of the high utilisation of WebServer
3	Rule 18	Component DB is redeployed on server1 from server3 because only one component DB was deployed on server3
4	Rule 5	Component CacheInfo is redeployed on server1 from server4 because component CacheInfo has the highest utilisation rate and interacts with component DB.
5	Rule 11	Component WebServer3 is replaced by its alternative component WebServer2 because of the highest utilisation of WebServer3
6	Rule 1	Component Scheduler is redeployed on server1 from server2 because of the highest utilisation of server2
7	Rule 1	Component DB is redeployed on server4 from server1 because of the highest utilisation of server1
8	Rule 3	The processing rate of server1 is increased to 19.9999 GHz from 10GHz because of the highest utilisation of server1
9	Rule 19	The processing rate of server2 is decreased to 7.8313 GHz from 10.0 GHz because of the lowest utilisation of server2
10	Rule 1	Component coreOnlineEngine is redeployed on server2 from server4 because of the highest utilisation of server4
11	Rule 3	The processing rate of server2 is increased to 14.7206 GHz from 7.8313 GHz because of the highest utilisation of server2
12	Rule 3	The processing rate of server4 is increased to 19.9940 GHz from 10GHz because of the highest utilisation of server4

Table 21
Mean values of metrics $avgNumRul_+$ and $avgNumMdf_+$ for PB_{cost} and MORE methods on the two problem instances of WebAPP and IES

	$avgNumRul_+$		$avgNumMdf_+$	
	WebApp	IES	WebApp	IES
PB_{cost}	9	8	34	79
MORE	6	5	16	35
the percentage of improvement of MORE relative to PB_{cost}	33%	38%	53%	56%

Table 22
Mean values of metrics $avgNumRul_+$ and $avgNumMdf_+$ for PB_{time} and MORE methods on the two problem instances of WebApp and IES

	$avgNumRul_+$		$avgNumMdf_+$	
	WebApp	IES	WebApp	IES
PB_{time}	9	8	26	79
MORE	6	5	17	35
the percentage of improvement of MORE relative to PB_{time}	33%	38%	35%	56%

The answer to RQ 4 shows that MORE can obtain better explanations for results by using fewer rules and modifying fewer architectural elements relatively. To further illustrate this point, we give the LQN models of the initial SA and the resulting SA obtained by PB_{cost} and PB_{time} for WebApp in Figures 23(a), (b), (c), respectively.

Figure 23 (d) presents the resulting LQN model obtained by MORE through the following three steps. Firstly, the solution set S is acquired by computing the intersection of sets η_{cost} and η_{time} (see Eq.(13)). Specifically, any one of S is better than the solutions obtained by PB_{cost} and PB_{time} with respect to the response time and cost. Secondly, the solution X^* with the shortest response time is picked out from S. Thirdly, the resulting LQN model, which is used to intuitively compare MORE with PB_{cost} and PB_{time}, is obtained through sequentially applying the modifications denoted by X^* into the initial LQN model. In Figure 23 (b), (c) and (d), the red elements were modified elements with respect to the initial LQN model. By counting these elements, we can observe that MORE modified fewer elements than PB_{cost} and PB_{time}. Furthermore, the sequences of rules corresponding to the three resulting LQN models obtained by PB_{cost}, PB_{time} and MORE are $< r2, r2, r2, r2, r2, r5, r3, r4 >$, $< r2, r2, r2, r2, r2, r3, r4, r2, r5 >$ and $< r5, r3, r2, r4, r2 >$, respectively. It is worth noting that, for comparison purposes, the rule numbers in the last sequence have been replaced according to mapping relation of rules between MORE and PB (PB_{cost} and PB_{time}). By comparing these sequences, we observe that MORE applied fewer rules

than PB_{cost} and PB_{time}. Thus, compared to PB_{cost} and PB_{time}, MORE can obtain shorter response time and lower cost, not only applying fewer rules but also modifying fewer elements.

Table 23
The p-values of metrics $avgNumRul_+$ and $avgNumMdf_+$ on the problem instances of WebApp and IES for comparing MORE against PB_{cost} and PB_{time} at the level of significance of 0.05

indicator	WebApp		IES	
	PB_{cost}	PB_{time}	PB_{cost}	PB_{time}
$avgNumRul_+$	1.2108E-12	1.2118E-12	1.2118E-12	1.2118E-12
$avgNumMdf_+$	1.2098E-12	1.2108E-12	1.2108E-12	1.2108E-12

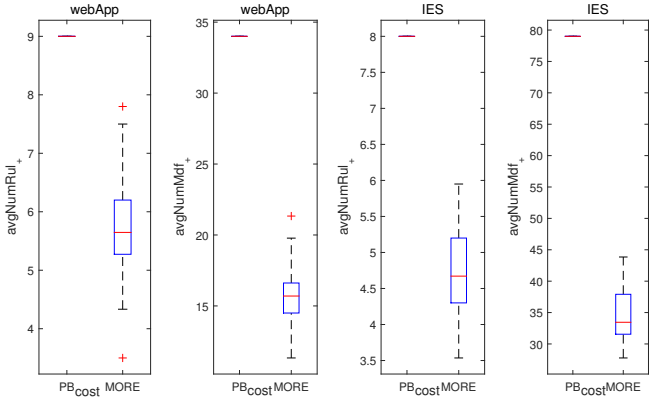


Figure 21: Boxplots using metrics $avgNumRul_+$ and $avgNumMdf_+$, to evaluate the explanations for the results obtained by PB_{cost} (left) and MORE (right) in the problem instances of WebApp (left) and IES (right)

In this section, MORE is applied to six problem instances with different scale and categories in order to answer from RQ1 to RQ4. The experimental results show that MORE is significantly better than PCM and PB on the quality of solutions and explanations for results. Therefore, MORE method has good usability.

6. Threats to validity

It is widely recognised that several factors can bias the validity of experimental studies. In this section, we discuss the validity of our study based on three types of threats, namely construct, internal, and external validity. Construct validity concerns the methodology employed to construct the experiment. Internal validity concerns possible bias in the way in which the results were obtained, while external validity concerns the possible bias of choice of experimental subjects.

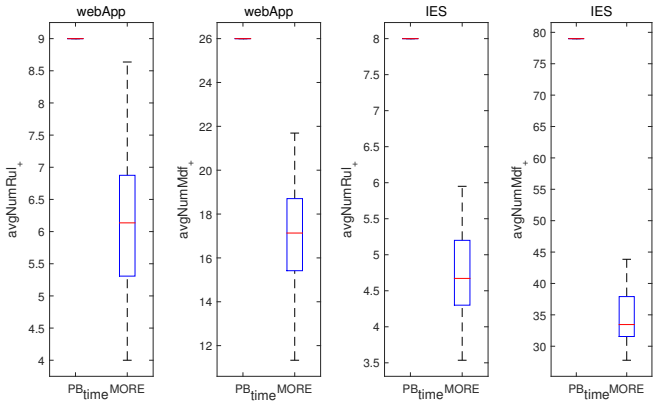
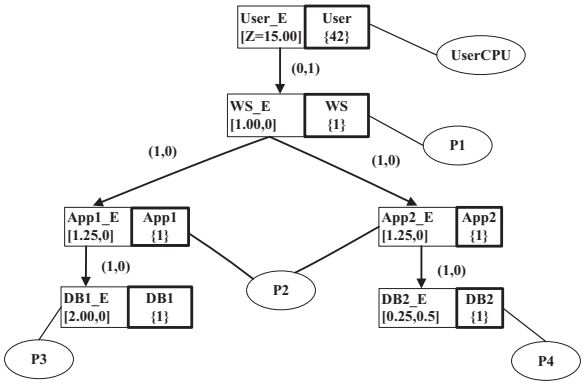


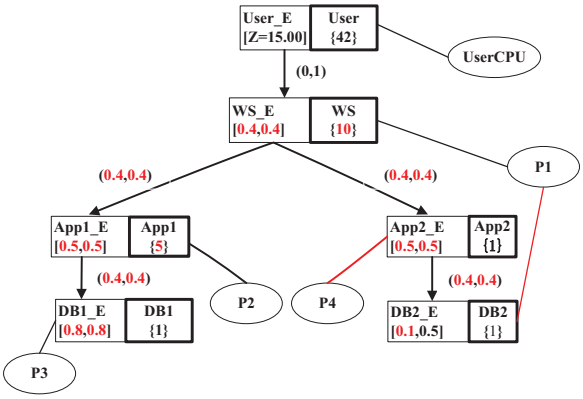
Figure 22: Boxplots using metrics $avgNumRul_+$ and $avgNumMdf_+$, to evaluate the explanations for the results obtained by PB_{time} (left) and MORE (right) in the problem instances of WebApp (left) and IES (right)

In our study, construct validity threats may arise from the choice of PB method and PCM method for the comparison with our method. In architecture-based performance optimisation, there are many rule-based methods and metaheuristic-based methods. However, PB and PCM are representative of these two kinds of methods, respectively. And they all used industrial problem instances and considered other quality attributes when performance is optimised. Another treat to construct validity can arise from the fact that our randomised search rules are not complete. For this threat, we have collected several architectural tactics for performance improvement and defined them by randomised search rules. Additional tactics will be collected in future work. Software architects can extend the randomised search rules in terms of requirements.

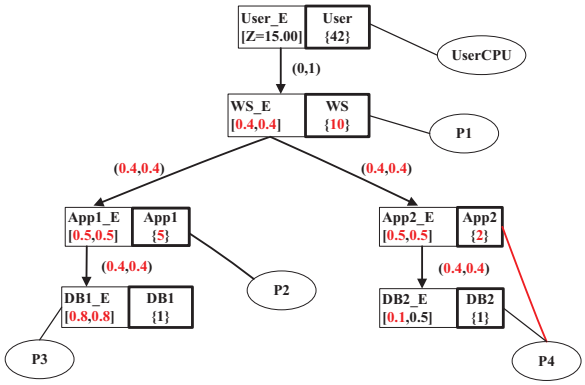
Internal validity threats can stem from the stochastic nature of the optimisation algorithms employed in our study. To mitigate these threats, we adopted the recommended practice for experimental studies in this research area (Arcuri and Briand, 2011). In particular, we reported results over 30 repeated runs of each experiment, and employed statistical tests to check for signif-



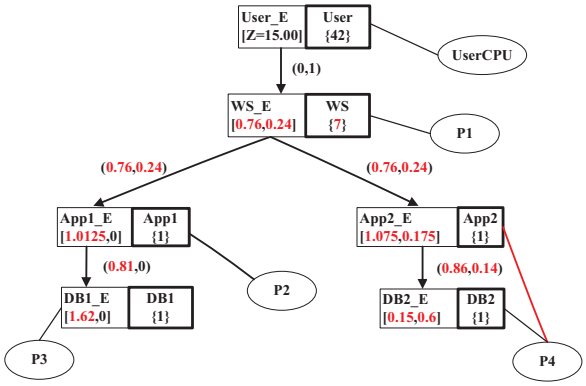
(a) The initial LQN model



(b) The resulting LQN model obtained by PB_{cost} with 21 modified elements compared to the initial model



(c) The resulting LQN model obtained by PB_{time} with 21 modified elements compared to the initial model



(d) The resulting LQN model obtained by MORE with 17 modified elements compared to the initial model

Figure 23: The resulting LQN models obtained by PB_{cost}, PB_{time} and MORE methods in comparison to the initial LQN model based on WebApp. Elements annotated in red are those modified architectural elements with respect to the initial ones.

icance on the achieved results. We used the Wilcoxon rank-sum tests to check for the statistical significance of the results achieved by the optimisation methods. As it is inadequate to merely show statistical significance alone, we used the Vargha-Delaney $\hat{A}_{12}$ metric for the effect size recommended by Arcuri and Briand (2011).

Our approach to external threats is also relatively standard in the empirical software engineering literature. Our experimental subjects are the applications which appear in the compared methods or are classical in the field of architecture-based performance optimisation. These applications range over the diverse domains and their scalability varies from small to larger industrial size. Despite all this, we cannot claim that our results generalise beyond these subjects studied.

7. Related work

Diagnosis and rule-based methods utilize practical knowledge of performance improvement when optimising SA. They can help architectural stakeholders to understand optimization results. While metaheuristic-based methods can find SAs with better performance by exploring larger optimisation space. The central contribution of this paper is to find good and explicable performance improvement solutions at the SA level by combining related methods, such as diagnosis methods, rule-based and metaheuristic-based optimisation methods. Here, we further discuss the state-of-the-art and

pinpoint some of their shortcomings.

7.1. Diagnosis methods

After many years of research and practice, a significant amount of performance improvement knowledge has been obtained from the activities of optimising performance. Smith and Williams (2003) systematically summarised the common performance problems in SA design as well as solutions to these problems. The 14 kinds of performance antipatterns are described in natural language by them. In order to automatically diagnose and improve performance flaws, many approaches for performance antipattern modelling, detection and solving have emerged in recent years.

When performance antipatterns are automatically detected and solved, the numerical parameters included in the precondition of performance antipatterns are represented as thresholds referring to either performance indices (e.g., a resource utilization) or design features (e.g., message traffic). However, Arcelli et al. (2013) has validated that these thresholds heavily affected detection and refactoring activities, and need to be set by heuristics. The thresholds in performance antipatterns were defined by the average number of performance indices or design features (Cortellessa et al., 2014; Sanctis, M. D. et al., 2017). Based on the specified thresholds, when multiple antipatterns are triggered, it will be left the architect to choose which antipattern to solve and then refactor SA. To mitigate the problems of threshold setting and antipattern selection, the detected performance antipatterns along with their occurrence probabilities (Trubiani et al., 2014) were proposed and applied in a fuzzy context where threshold values cannot be determined, but only their lower and upper bounds were known.

Although the tool provided by the diagnosis methods can obtain a list of the detected performance antipatterns that may cause performance problems, architects often need to manually do three tasks: choose an antipattern from the list, choose an action supposed to be the best one from refactoring actions given by the selected antipattern, and apply this action into SA to improve performance. The diagnosis process does not stop until the software architects remove all the found antipatterns or until performance requirements are met.

7.2. Rule-based methods

Rule-based methods provided the ability to automatically apply performance improvement knowledge from antipatterns or design tactics into SA in the form of rules. These methods used optimisation algorithms to obtain good SAs by combining the predefined rules.

The ArchE framework (Mcgregor et al., 2007) was proposed to support the software designers in creating architectures that meet quality requirements. It embodies knowledge of quality attributes and the relation between the achievement of quality requirements and architectural design. However, the suggestions (or tactics) are not well explained, and it is not clear to which extent the approach can be applied. An approach to optimise deployment and configuration decisions in the context of distributed, real-time, and embedded component-based systems (Kavimandan and Gokhale, 2009) was presented. Enhanced bin packing algorithms and schedulability analysis have been used to make fine-grained assignments of components to different middleware containers, since they are known to impact the system's performance and resource consumption. However, the scope of this approach is limited to deployment and configuration features.

PB method (Xu, 2012) adopted tree-based search algorithm to optimise performance and cost. During optimisation, each rule is used in a fixed order determined by the predefined search strategies. PB can explain how to obtain the resulting SA from the initial SA by the best search path of rule applications. However, due to the predefined search strategies, the search space is limited and suboptimal solutions may be obtained. Our previous works (Du et al., 2015b,a) introduced metaheuristic algorithms into PB method to search larger performance improvement space. In one paper (Du et al., 2015b), GA was proposed to optimise performance and the adaptive mutation operator was designed by considering history information of rule applications to speed up the algorithm run. Except for performance, the number of rules with improvement

effect was also considered as a goal and optimised by NSGA-II in another paper (Du et al., 2015a).

Based on performance antipatterns expressed as first-order logics , Arcelli et al. (2018b) implemented a set of antipattern rules on UML. And these rules were used in tree-based search algorithm to explore SA optimisation space for finding the solutions with better performance. Furthermore, in EASIER approach (Arcelli et al., 2018a), antipattern rules were also implemented on Æmilia ADL and applied by custom NSGA-II algorithm to optimise SA with respect to performance, the intensity of changes, and the number of performance antipatterns occurring in the SA. However, there are no industrial problem instances investigated in the both papers (Arcelli et al., 2018b,a).

In rule-based methods, threshold in the precondition and improvement amplitude in the action of each used rule need to be set before optimisation by software architects. Setting an appropriate threshold is rather hard because it needs a deep understanding of performance improvement knowledge and SA to be optimized in hand. At the same time, the explanation with threshold is not particularly convincing and reliable for architectural stakeholders. What's more, in each used rule, the predefined threshold and the amplitude can potentially prevent optimisation algorithms from searching larger space. Among existing rule-based methods, some methods (Arcelli et al., 2018b,a) have not been validated by industrial problem instances, other methods (Arcelli et al., 2018b,a; Du et al., 2015a,b) have not considered cost when optimisating performance. Therefore, PB method is regarded as a suitable compared method in this paper.

7.3. Metaheuristic-based methods

Based on the different ADLs, such as PCM, UML and East-AADL, several metaheuristic approaches for performance optimisation have been proposed in terms of LQN or queueing network, which are popular performance models. To automatically find the SAs with better performance, these approaches use metaheuristic algorithms to search the design space, which is determined by a few DoFs, such as component selection, component allocation and resource configuration. Some methods take the performance as a single quality attribute to be optimised. Others are used to optimise performance and other quality attributes such as reliability and safety.

7.3.1. Performance as a single quality attribute to be optimised

Response time, utilisation, throughput are considered as the performance indices to be optimised independently or balanced against design constraint (i.e. cost) (Li et al., 2010; Amoozegar, 2015; Li et al., 2011) during optimisation.

Based on PCM, Martens and Koziolek (2009) applied steepest-ascent hill-climbing algorithm to optimise response time. Li et al. (2010) proposed a kind of performance model for SAP ERP application based on PCM and LQN, and then used S-Metric Selection Evolutionary Multi-objective Optimisation Algorithm to optimise response time and hardware cost. Tribastone (2014) defined a fluid model of the closed queuing network with generalised processor sharing service. Based on this model, they used a genetic algorithm to find the best tradeoff between throughput and cost for a canonical three-layered SA.

Based on UML with MARTE, Amoozegar (2015) proposed a multi-objective gravitational search algorithm to optimise response time and hardware costs. Based on East-AADL, Li et al. (2011) applied evolutionary multi-objective optimisation algorithms, such as NSGA-II and SPEA2, to optimise data flow latency, processor utilisation and cost. Their experimental results indicate that the evolutionary multi-objective algorithms are suitable for large problems.

7.3.2. Optimising performance and other quality attributes

In several studies, based on different ADLs, other quality attributes or design constraints (i.e. cost) are also regarded as optimisation objectives along with performance in order to find the better tradeoffs between a set of quality attributes.

Based on PCM, Martens et al. (2009) presented

multi-criteria genetic algorithm to optimise performance and reliability. Moreover, Martens et al. used NSGA-II to optimise performance, dependability and costs (Martens and Koziolek, 2009), to optimise performance, availability, and cost (Martens et al., 2010a; Koziolek et al., 2013) and to optimise performance, reliability and cost (Martens et al., 2010b; Reussner, 2010). Furthermore, SMDE4PO (Du et al., 2017) and PerOpteryx methods (Koziolek et al., 2011a; Koziolek, 2014) were proposed to optimise performance and cost. SMDE4PO (our previous work) introduced surrogate model into multi-objective differential evolutionary algorithm to reduce computational cost incurred by performance evaluation. PerOpteryx method used rules to describe design tactics and integrated these rules into NSGA-II as heuristic operators to speed up search for good candidates. However, these rules contain the thresholds in precondition and the improvement amplitudes in action. When rules as heuristic operators are applied, and the preconditions of multiple rules are met, PerOpteryx generates multiple candidates. To decide for one candidate, PerOpteryx also requires architects to set the weight of each rule. Meanwhile, in PerOpteryx method, an individual is encoded as DoFI and not related to rules. Not only rules but also crossover and mutation operators can be used to generate candidates. As a result, PerOpteryx cannot explain how to obtain one candidate by only use of rules.

Based on AADL, Meedeniya et al. (2012) proposed a framework tool Archaeopteryx which supports NSGA-II to optimise performance and reliability of the embedded system. NSGA-II (Etemaadi and Chaudron, 2012), SPEA2 (Etemaadi et al., 2013) and SMS-EMOA (Ramin, 2014) are used to optimise performance, safety and cost. To enlarge the search space, Etemaadi further proposed architecture topology and load balancing as two novel DoFs (Etemaadi and Chaudron, 2015). Their works are remarkable and can improve system architecture design and find new architectural solutions. Rahmoun et al. (2015, 2017) used model transformation rules to implement the modifications for DoFI in SA, and NSGA-II to search the sequences of rule applications while optimising performance and reliability. Although Rahmoun's method defined the model transformation rules to modify DoFI, these rules cannot be used to describe performance improvement knowledge due to the shortage of performance information. Performance information is only acquired by performance evaluation and not included in SA. As a result, Rahmoun's method cannot explain the cause of transformation.

Based on UML, SQME (Sedaghatbaf and Azgomi, 2019) was proposed to automatically optimise SA with respect to performance, reliability and cost by using NSGA-II algorithm. Specially, this method introduced the belief and plausibility measures to define a new dominance relation, considering the uncertainty of parameters (e.g., workload and resource demands). However, during architecture optimization process in the SQME, the uncertain parameters need to be estimated by domain experts. Therefore, the value of the parameters are possibly biased so as to do harm to quality of optimisation results. In addition, the problem instance that they investigated was rather small.

Most of metaheuristic-based methods ignore the performance improvement knowledge represented by performance antipatterns and architectural tactics. To some extent, the SA is randomly modified based on DoFIs. And the optimisation results can be only explained according to the differences between the initial SA and the resulting SA. We call that metaheuristic-based methods have the trivial explicability considering that the causes of modifications are absent. The disadvantage on trivial explicability is that it is hard for architectural stakeholders to understand the optimisation results and select SAs when trading off other quality attributes. Among a variety of metaheuristic-based methods, PCM method is selected as compared method, two reasons are given below. Firstly, PCM method is a hybrid multi-objective performance optimisation method. The experimental results show that PCM can obtain better solutions than the pure evolutionary optimization method that only uses an original multi-objective evolutionary algorithm. Secondly, the

cases in PCM have different scale and coverA various categories so that they are enough to support our comparative experiments.

8. Conclusions

To address the challenges of limited search space in rule-based methods and lacking explanations for results obtained in metaheuristic-based methods, we presented an approach named MORE to performance optimisation at the SA level. First, we defined MORE-R which do not need preset thresholds and action amplitudes. As a result, MORE-R not only are easily used by software architects but also provide explanation without parameters. Then, MORE-P was presented by fully considering the relationship between all possible composite applications of rules and each optimisation goals. Furthermore, we designed MORE-EA to solve the MORE-P by introducing a do-nothing rule and proposing evolutionary operators with repair mechanism. The experimental results show that MORE can obtain more explicable and higher quality solutions than those of PCM and PB methods.

Considering the complexity introduced by computing objective value in MORE, we will introduce the predictive models to reduce time consuming of solving objective value so that MORE will obtain better scalability in the future. Additionally, more DoFs and randomised search rules will be also considered.

Acknowledgments

This work is supported by the Royal Society International Exchanges (IE151226), Talent support program of High school in the new century of Fujian Province (Year 2017), the Natural Science Foundation of Fujian Province (Nos.2020J01165, 2017J01498), the Natural Science Foundation of Hubei Province (No. 2018CFB689).

References

Aleti, A., Buhnova, B., Grunske, L., Koziolek, A. , Meedeniya, I. , 2013. Software Architecture Optimization Methods: A Systematic Literature Review. IEEE Transactions on Software Engineering 39, 658–683. doi:10.1109/TSE.2012.64.

Amoozegar, M.e.a., 2015. A multi-objective approach to model-driven performance bottlenecks mitigation. Scientia Iranica. Transaction D, Computer Science & Engineering, Electrical 22, 1018–1030.

Arcelli, D., Cortellessa, V., D'Emidio, M., Di Pompeo, D., 2018a. EASIER: An Evolutionary Approach for multi-objective Software archItecturE Refactoring, in: 2018 IEEE International Conference on Software Architecture (ICSA), IEEE, Seattle, USA. pp. 105–10509.

Arcelli, D., Cortellessa, V., Di Pompeo, D., 2018b. Performance-driven software model refactoring. Information and Software Technology 95, 366–397.

Arcelli, D., Cortellessa, V., Trubiani, C., Wu, W., Margaria, T., Padberg, J., Taentzer, G., 2013. Experimenting the influence of numerical thresholds on model-based detection and refactoring of performance antipatterns, in: ECEASST.

Arcuri, A., Briand, L., 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering, in: Proceedings of the 33rd International Conference on Software Engineering, ACM, Waikiki, Honolulu, HI, USA. pp. 1–10.

Becker, S., Koziolek, H., Reussner, R., 2009. The Palladio component model for model-driven performance prediction. Journal of Systems and Software 82, 3–22. doi:10.1016/j.jss.2008.03.066.

Boehm, B., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J., Steece, B., 2009. Cost Estimation with COCOMO II. 1st ed., Prentice Hall Press, Upper Saddle River, NJ, USA.

Brosch, F., Buhnova, B., Koziolek, H., Reussner, R., 2011. Reliability prediction for fault-tolerant software architectures, in: Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS, ACM, Boulder, Colorado, USA. pp. 75–84.

Brosch, F., Koziolek, H., Buhnova, B., Reussner, R., 2012. Architecture-based reliability prediction with the palladio component model. Software Engineering, IEEE Transactions on 38, 1319–1339.

Brosig, F., Meier, P., Becker, S., Koziolek, A., Koziolek, H., Kounev, S., 2015. Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-Based Architectures. IEEE Transactions on Software Engineering 41, 157–175. doi:10.1109/TSE.2014.2362755.

Chen, Z., Chen, B., Xiao, L., Wang, X., Chen, L., Liu, Y., Xu, B., 2018. Speedoo: Prioritizing performance optimization opportunities, in: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), IEEE. pp. 811–821.

Cortellessa, V., Marco, A.D., Trubiani, C., 2014. An approach for modeling and detecting software performance antipatterns based on first-order logics. Software & Systems Modeling 13, 391–432.

Deb, K., 2011. Multi-objective optimisation using evolutionary algorithms: an introduction, in: Multi-objective evolutionary optimisation for product design and manufacturing. Springer, pp. 3–34.

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

Distefano, S., Scarpa, M., Puliafito, A., 2011. From UML to Petri Nets: The PCM-Based Methodology. IEEE Transactions on Software Engineering 37, 65–79. doi:10.1109/TSE.2010.10.

Du, X., Ni, Y., Wu, X., Ye, P., Yao, X., 2017. Surrogate Model Assisted Multi-objective Differential Evolution Algorithm for Performance Optimization at Software Architecture Level, in: Asia-Pacific Conference on

Simulated Evolution and Learning.(SEAL 2017), Springer, Shenzhen, China. pp. 334–346.

Du, X., Ni, Y., Ye, P., 2015a. A Multi-objective Evolutionary Algorithm for Rule-based Performance Optimization at Software Architecture Level, in: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, ACM, Madrid, Spain. pp. 1385–1386. doi:10.1145/2739482.2764705.

Du, X., Yao, X., Ni, Y., Minku, L.L., Ye, P., Xiao, R., 2015b. An evolutionary algorithm for performance optimization at software architecture level, in: 2015 IEEE Congress on Evolutionary Computation (CEC), IEEE, Sendai, Japan. pp. 2129–2136. doi:10.1109/CEC.2015.7257147.

Durillo, J.J., Nebro, A.J., 2011. jMetal: A Java framework for multi-objective optimization. Advances in Engineering Software 42, 760–771.

Eiben, A.E., Smith, J.E., 2003. Introduction to Evolutionary Computing. Natural Computing Series, Springer Berlin Heidelberg.

Etemaadi, R., Chaudron, M.R., 2012. Varying topology of component-based system architectures using metaheuristic optimization, in: Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference On, IEEE, Cesme, Izmir, Turkey. pp. 63–70.

Etemaadi, R., Chaudron, M.R., 2015. New degrees of freedom in metaheuristic optimization of component-based systems architecture: Architecture topology and load balancing. Science of Computer Programming 97, 366–380.

Etemaadi, R., Lind, K., Heldal, R., Chaudron, M.R., 2013. Quality-driven optimization of system architecture: Industrial case study on an automotive sub-system. Journal of Systems and Software 86, 2559–2573.

Fonseca, C.M., Knowles, J.D., Thiele, L., Zitzler, E., 2005. A tutorial on the performance assessment of stochastic multiobjective optimizers, in: Third International Conference on Evolutionary Multi-Criterion Optimization, Springer, Guanjuato,Mexico. p. 240.

Gokhale, S.S., 2007. Architecture-Based Software Reliability Analysis: Overview and Limitations. IEEE Transactions on Dependable and Secure Computing 4, 32. doi:http://dx.doi.org/10.1109/TDSC.2007.4.

Grissom, R.J., Kim, J.J., 2005. Effect Sizes for Research: A Broad Practical Approach. Lawrence Earlbaum Associates.

Immonen, A., Niemelä, E., 2008. Survey of reliability and availability prediction methods from the viewpoint of software architecture. Software and Systems Modeling 7, 49–65. doi:10.1007/s10270-006-0040-x.

ISO, 2011. COSMIC International Standard(ISO/IEC 19761:2011).

Jong, K.D., 2016. Evolutionary computation: a unified approach, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, pp. 185–199.

Kavimandan, A., Gokhale, A., 2009. Applying Model Transformations to Optimizing Real-Time QoS Configurations in DRE Systems, in: Mirandola, R., Gorton, I., Hofmeister, C. (Eds.), Architectures for Adaptive Software Systems: 5th International Conference on the Quality of Software Architectures, QoSA 2009, East Stroudsburg, PA, USA, June 24-26, 2009 Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 18–35. doi:10.1007/978-3-642-02351-4_2.

Kounev, S., 2006. Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. IEEE Transactions on Software Engineering 32, 486–502. doi:10.1109/TSE.2006.69.

Koziolek, A., 2014. Automated Improvement of Software Architecture Models for Performance and Other Quality Attributes. volume 7 of *The Karlsruhe Series on Software Design and Quality*. KIT Scientific Publishing.

Koziolek, A., Ardagna, D., Mirandola, R., 2013. Hybrid multi-attribute QoS optimization in component based software systems. Journal of Systems and Software 86, 2542–2558.

Koziolek, A., Koziolek, H., Reussner, R., 2011a. Peropteryx: Automated application of tactics in multi-objective software architecture optimization, in: Proceedings of the Joint ACM SIGSOFT Conference–QoSA and ACM SIGSOFT Symposium–ISARCS on Quality of Software Architectures–QoSA and Architecting Critical Systems–ISARCS, ACM, Boulder, Colorado, USA. pp. 33–42.

Koziolek, H., Schlich, B., Bilich, C., Weiss, R., Becker, S., Krogmann, K., Trifu, M., Mirandola, R., Koziolek, A., 2011b. An industrial case study on quality impact prediction for evolving service-oriented software, in: Proceedings of the 33rd International Conference on Software Engineering, IEEE, Honolulu, HI, USA. pp. 776–785.

Li, H., Casale, G., Ellahi, T., 2010. SLA-driven planning and optimization of enterprise applications, in: Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering, ACM, San Jose, California, USA. pp. 117–128.

Li, R., Etemaadi, R., Emmerich, M.T., Chaudron, M.R., 2011. An evolutionary multiobjective optimization approach to component-based software architecture design, in: Evolutionary Computation (CEC), 2011 IEEE Congress On, IEEE, New Orleans, LA, USA. pp. 432–439.

Martens, A., Ardagna, D., Koziolek, H., Mirandola, R., Reussner, R., 2010a. A Hybrid Approach for Multi-attribute QoS Optimisation in Component Based Software Systems, in: Heineman, G.T., Kofron, J., Plasil, F. (Eds.), Research into Practice – Reality and Gaps: 6th International Conference on the Quality of Software Architectures, QoSA 2010, Prague, Czech Republic, June 23 - 25, 2010. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 84–101. doi:10.1007/978-3-642-13821-8_8.

Martens, A., Brosch, F., Reussner, R., 2009. Optimising multiple quality criteria of service-oriented software architectures, in: Proceedings of the 1st International Workshop on Quality of Service-Oriented Software Systems, ACM, Amsterdam, The Netherlands. pp. 25–32.

Martens, A., Koziolek, H., 2009. Automatic, model-based software performance improvement for component-based software designs. Electronic Notes in Theoretical Computer Science 253, 77–93.

Martens, A., Koziolek, H., Becker, S., Reussner, R., 2010b. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms, in: Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering, ACM, California, USA. pp. 105–116.

Mcgregor, J.D., Bachmann, F., Bianco, P., Klein, M., 2007. Using ArchE in the Classroom: One Experience. TECHNICAL NOTE CMU/SEI-2007-TN-001. Carnegie Mellon University.

Meedeniya, I., Aleti, A., Avazpour, I., Amin, A., 2012. Robust archeopterix: Architecture optimization of embedded systems under uncertainty, in: Software Engineering for Embedded Systems, 2012 2nd International Workshop On, IEEE, Zurich, Switzerland. pp. 23–29.

Meier, J.D., Vasireddy, S., Babbar, A., Mackman, A., 2004. Improving. NET Application Performance and Scalability. Microsoft Corporation.

Meunier, H., Talbi, E.G., Reininger, P., 2000. A multiobjective genetic algorithm for radio network optimization, in: Proceedings of the 2000 Congress on Evolutionary Computation. CEC, IEEE, La Jolla, CA, USA. pp. 317–324.

Minku, L.L., Yao, X., 2014. How to Make Best Use of Cross-company Data in Software Effort Estimation?, in: Proceedings of the 36th Inter-

national Conference on Software Engineering, ACM, Hyderabad, India. pp. 446–456. doi:10.1145/2568225.2568228.

Navarro, E., Cuesta, C.E., Perry, D.E., González, P., 2013. Antipatterns for architectural knowledge management. International Journal of Information Technology & Decision Making 12, 547–589.

OMG, 2017. UML Superstructure Specification, v2.1.1.

Poort, E., Vliet, E.V.D., 2015. Architecting in a Solution Costing Context: Early Experiences with Solution-Based Estimating, in: Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference On, IEEE, Montreal, QC, Canada. pp. 127–130.

Rahmoun, S., Borde, E., Pautet, L., 2015. Automatic selection and composition of model transformations alternatives using evolutionary algorithms, in: Proceedings of the 2015 European Conference on Software Architecture Workshops, ACM, Dubrovnik, Cavtat, Croatia. p. Article No. 25.

Rahmoun, S., Mehiaoui-Hamitou, A., Borde, E., Pautet, L., Soubiran, E., 2017. Multi-objective exploration of architectural designs by composition of model transformations. Software & Systems Modeling , 1–21.

Ramin, E.I., 2014. Quality-Driven Multi-Objective Optimization of Software Architecture Design: Method, Tool, and Application. Ph.D. thesis. Leiden Institute of Advanced Computer Science (LIACS), Faculty of Science, Leiden University.

Reussner, R., 2010. Domain-Specific Heuristics for Automated Improvement of PCM-Based Architectures. Ph.d. dissertation. University of Karlsruhe. Karlsruhe, Germany.

Salcedo-Sanz, S., 2009. A survey of repair methods used as constraint handling techniques in evolutionary algorithms. Computer science review 3, 175–192.

Sanctis, M. D., Trubiani, C. , Cortellessa, V. , Marco, A. D., Flamminj, M., 2017. A model-driven approach to catch performance antipatterns in adl specifications. Inf. Softw. Technol. 83, 35–54.

Sarro, F., Petrozziello, A., Harman, M., 2016. Multi-objective Software Effort Estimation, in: Proceedings of the 38th International Conference on Software Engineering, ACM, Austin, Texas, USA. pp. 619–630. doi:10.1145/2884781.2884830.

Sedaghatbaf, A., Azgomi, M.A., 2019. SQME: A framework for modeling and evaluation of software architecture quality attributes. Software & Systems Modeling 18, 2609–2632.

Shepperd, M., Schofield, C., 1997. Estimating Software Project Effort Using Analogies. IEEE Trans. Softw. Eng. 23, 736–743. doi:10.1109/32.637387.

Slot, R., 2010. A Method for Valuing Architecture-Based Business Transformation and Measuring the Value of Solutions Architecture. Ph.d. dissertation. University of Amsterdam. Amsterdam, The Netherlands.

Smith, C.U., Williams, L.G., 2003. More new software performance antipatterns: Even more ways to shoot yourself in the foot, in: Computer Measurement Group Conference, pp. 717–725.

Software Design and Quality Group, Karlsruhe Institute of Technology, . STE. https://sdqweb.ipd.kit.edu/eclipse/palladio/examples/releases/4.1.0/SimpleHeuristicsExample_Example.zip. 2017.

Srinivas, N., Deb, K., 1994. Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. Evolutionary Computation 2, 221–248. doi:10.1162/evco.1994.2.3.221.

Tribastone, M., 2013. A fluid model for layered queueing networks. IEEE Transactions on Software Engineering 39, 744–756. doi:10.1109/TSE.2012.66.

Tribastone, M., 2014. Efficient optimization of software performance models via parameter-space pruning, in: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, ACM, Dublin, Ireland. pp. 63–73.

Tribastone, M., Gilmore, S., Hillston, J., 2012. Scalable Differential Analysis of Process Algebra Models. IEEE Transactions on Software Engineering 38, 205–219. doi:10.1109/TSE.2010.82.

Trubiani, C., Koziolek, A., Cortellessa, V., Reussner, R., 2014. Guilt-based handling of software performance antipatterns in palladio architectural models. Journal of Systems and Software 95, 141–165.

Xu, J., 2012. Rule-based automatic software performance diagnosis and improvement. Performance Evaluation 69, 525–550. doi:10.1016/j.peva.2009.11.003.

Zitzler, E., Thiele, L., 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE transactions on Evolutionary Computation 3, 257–271.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Da Fonseca, V.G., 2003. Performance assessment of multiobjective optimizers: An analysis and review. Evolutionary Computation, IEEE Transactions on 7, 117–132.

Youcong Ni is currently an Associate Professor at the College of Mathematics and Informatics, Fujian Normal University(CN). He received his PhD in Software Engineering from State Key Laboratory of Software Engineering, Wuhan University(CN) in 2010. He was a research visitor in the University College London for one year. His research interests include Search Based Software Engineering (SBSE) and evolutionary computation. He has published over 30 papers including Swarm and Evolutionary Computation, Soft Computing, GECCO and CEC.

Xin Du is currently a Professor at the College of Mathematics and Informatics, Fujian Normal University(CN). She received her PhD in Software Engineering from State Key Laboratory of Software Engineering, Wuhan University(CN) in 2010. She was a research visitor in the University of Birmingham for one year. Her research interests include Search Based Software Engineering (SBSE) and intelligent computation. Her work has been published over 30 papers including IEEE Internet of Things Journal, Swarm and Evolutionary Computation, Soft Computing, GECCO and CEC.

Peng Ye is currently an Associate Professor at College of Mathematics and Computer, Wuhan Textile University (CN). He received his PhD in Software Engineering from State Key Laboratory of Software Engineering, Wuhan University(CN) in 2009. His research interests include search based software architecture optimisation and evolutionary computation. He has published over 20 papers including GECCO and CEC international conference.

Leandro L. Minku is a Lecturer in Intelligent Systems at the School of Computer Science, University of Birmingham (UK). Prior to that, he was a Lecturer in Computer Science at the University of Leicester(UK), and a research fellow at the University of Birmingham (UK). He received the PhD degree in Computer Science from the University of Birmingham (UK) in 2010. Dr. Minku's main research interests are machine learning for software engineering, search-based software engineering, machine learning for non-stationary environments / data stream mining, and ensembles of learning machines. His work has been published in internationally renowned journals such as IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology, IEEE Transactions on Knowledge and Data Engineering, and IEEE Transactions on Neural Networks and Learning Systems.

Xin Yao is a Chair Professor of Computer Science at the Southern University of Science and Technology, Shenzhen, China, and a part-time Professor of Computer Science at the University of Birmingham, UK. He is an IEEE Fellow, and a Distinguished Lecturer of IEEE Computational Intelligence Society (CIS). His major research interests include evolutionary computation, ensemble learning, and their applications in software engineering.

Mark Harman is currently an engineering manager at Facebook and a part time professor of Software Engineering in the Department of Computer Science at University College London, where he directed the CREST centre for ten years (2006-2017) and was Head of Software Systems Engineering (2012-2017). He is widely known for work on source code analysis, software testing, appstore analysis and Search Based Software Engineering (SBSE), a field he co-founded and which has grown rapidly to include over 1,600 authors spread over more than 40 countries.

Ruliang Xiao received the Ph.D. degree in computer software and theory from Wuhan University,China,in 2007. He is currently a Professor with the College of Mathematics and Informatics, Fujian Normal University,China.His research interests include system security engineering and computing intelligence. He has authored three books,over 20 patents for invention,and published over 50 papers in international journals and conference proceedings. He was a recipient of the Fujian Provincial Science and Technology Progress Award in 2016.