Multi-Objective Approaches to Optimal Testing Resource Allocation in Modular Software Systems

Zai Wang, *Student Member, IEEE*, Ke Tang, *Member, IEEE*, and Xin Yao, *Fellow, IEEE*

***Abstract*—Software testing is an important issue in software engineering. As software systems become increasingly large and complex, the problem of how to optimally allocate the limited testing resource during the testing phase has become more important, and difficult. Traditional Optimal Testing Resource Allocation Problems (OTRAPs) involve seeking an optimal allocation of a limited amount of testing resource to a number of activities with respect to some objectives (e.g., reliability, or cost). We suggest solving OTRAPs with Multi-Objective Evolutionary Algorithms (MOEAs). Specifically, we formulate OTRAPs as two types of multi-objective problems. First, we consider the reliability of the system and the testing cost as two objectives. Second, the total testing resource consumed is also taken into account as the third objective. The advantages of MOEAs over state-of-the-art single objective approaches to OTRAPs will be shown through empirical studies. Our study has revealed that a well-known MOEA, namely Nondominated Sorting Genetic Algorithm II (NSGA-II), performs well on the first problem formulation, but fails on the second one. Hence, a Harmonic Distance Based Multi-Objective Evolutionary Algorithm (HaD-MOEA) is proposed and evaluated in this paper. Comprehensive experimental studies on both parallel-series, and star-structure modular software systems have shown the superiority of HaD-MOEA over NSGA-II for OTRAPs.**

***Index Terms*—Multi-objective evolutionary algorithm, parallel-series modular software system, software engineering, software reliability, software testing, star-structure modular software system.**

ACRONYMS

OTRAP	optimal testing resource allocation problem
SRGM	software reliability growth model
MOEA	multi-objective evolutionary algorithm
NSGA-II	nondominated sorting genetic algorithm II
HaD-MOEA	harmonic distance based multi-objective evolutionary algorithm

NOTATIONS

T	total testing resource (or time)
T_i	testing resource allocated to module i
T_i^*	optimal allocation resource to module i
C	total testing cost
R	system reliability
$C_i(R_i)$	cost function of module i according to the reliability of module i (R_i)
$m(t)$	mean value function in NHPP
$R_i(x\|T_i)$	reliability of module i after a testing period T_i. x is a constant number associated with the studied system
T_{li}	testing resource allocated to the ith modular in the lth parallel group
T_j	testing resource allocated to the jth serial module
R_{li}	reliability of the ith module in the lth parallel group
R_j	reliability of the jth serial module
T_C	testing resource allocated to the central unit in star-structure modular software system

Manuscript received March 22, 2009; revised January 18, 2010 and March 17, 2010; accepted March 17, 2010 Date of publication August 16, 2010; date of current version September 01, 2010. . This work was supported in part by the National Natural Science Foundation of China under Grant 60802036 and Grant U0835002, by the Engineering and Physical Sciences Research Council (EPSRC Grant EP/D052785/1), and by the Fund for Foreign Scholars in University Research and Teaching Programs (111 project) in China under Grant B07033. Associate Editor: G. Levitin.

Z. Wang and K. Tang are with the Nature Inspired Computation and Applications Laboratory (NICAL), School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China (e-mail: wangzai@mail.ustc.edu.cn; ketang@ustc.edu.cn).

X. Yao is with the Nature Inspired Computation and Applications Laboratory (NICAL), School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China, and also with CERCIA, School of Computer Science, University of Birmingham, Edgbaston, Birmingham, B15 2TT, U.K. (e-mail: x.yao@cs.bham.ac.uk).

Digital Object Identifier 10.1109/TR.2010.2057310

I. INTRODUCTION

A software development process typically consists of four phases [1]: specification, designing, coding, and testing. The requirements of consumers are defined in the specification phase. After that, the structures and details are designed and implemented during the next two phases. Finally, the software systems are tested to detect and correct latent software errors during the testing phase. The testing phase, which aims to improve the reliability of a software system, is the most costly, time-consuming phase among the four phases [1]. About half of the resources consumed during the software development cycle are testing resources [1]. Moreover, because the sizes of software systems have increased significantly during the past decades,

effective utilization of limited testing resource has become even more important than before.

A software system is typically comprised of a number of modules. Each module needs to be assigned appropriate testing resources before the testing phase. Hence, a natural question is how to allocate the testing resources to the modules so that the reliability of a software system is maximized. Such a problem was formally defined by Ohtera and Yamada as the Optimal Testing Resource Allocation Problems (OTRAPs) [1]. Although testing resources can be allocated in rather simple ways (e.g. average allocation, random allocation, and proportional allocation), Huo *et al.* [2] proved that an optimal allocation scheme may lead to significant improvement in terms of the reliability of a software system. In other words, it is well worth optimizing the allocation scheme.

Solving OTRAPs is a non-trivial task. Much effort has been devoted to this topic since the 1990s [1]–[18], and progress has been made in the way of either proposing more precise/practical formulations of OTRAPs, or utilizing novel problem-solving techniques. We start revisiting the literature from the former type of work. An OTRAP is typically concerned with three factors: reliability, cost, and testing resources. To explicitly formulate an OTRAP, the relationship between these factors needs to be precisely defined. In the literature, the relationship between reliability and testing resources was usually formulated by Software Reliability Growth Models (SRGMs), where the reliability is usually some metric of the failure data, such as the number of failures, time of occurrence, failure severity, or the interval between two consecutive failures [13]. The SRGMs describe reliability growth during software development processes, and can be viewed as formulating the reliability of a software system as a function of the testing resource allocation scheme. Unsurprisingly, early work exclusively aimed at maximizing the reliability with a given budget of resource, by means of minimizing the remaining errors [1], [4], [5], minimizing the number of software faults detected [2], or directly maximizing a function that quantifies system reliability [6]–[8], [10]. More recently, testing cost is attracting more attention. Intuitively speaking, testing cost measures the cost required for attaining a given level of reliability. It is essentially a function of reliability, and thus also a function of testing resources [11]–[13], [15]–[18]. Recent work on OTRAPs either proposed minimizing testing cost instead of maximizing reliability, or incorporated additional constraints on the testing cost (e.g., the reliability is maximized subject to given bounds of the testing cost and resource) [12], [13], [15]–[18].

A lot of traditional optimization approaches have been employed to solve OTRAPs, such as nonlinear programming [2]–[5], [13], [16], gradient projection method [6], and dynamic programming [5]. However, these approaches only guarantee local convergence. They may be easily trapped in local optima if the solution space of an OTRAP is multimodal. Although such disadvantage can be alleviated by running the algorithms multiple times from different initial solutions, choosing appropriate initial solutions requires one to be familiar with both the algorithm, and the characteristics of OTRAPs, which is usually very difficult for ordinary users. Alternatively, evolutionary algorithms are meta-heuristics that possess the strong capability of global search, and are usually not very sensitive to initial solutions. Their effectiveness has been demonstrated on a large spectrum of problems in the reliability optimization field, such as resource management and task partition in grid systems [19]–[21], redundancy allocation [22], [23], reliability optimization of weighted voting systems [24], and OTRAPs [7]–[12], [15], [17], [18]. In particular, evolutionary algorithms have been reported to perform better than some other techniques on OTRAPs [7], [9]–[12], [15], [17], [18].

Being intensively investigated in the past decade, OTRAPs were mostly handled as optimization problems with a single objective, i.e., the testing resource was allocated with the only purpose to maximize the system reliability, or minimize the testing cost. However, both reliability and testing cost are important for software development, and it is unrealistic to overlook either of them. Unfortunately, given a budget of testing resources, more testing cost is usually inevitable if we want to improve the reliability of a software system. Hence, it is impossible that a single solution is optimal in terms of both reliability and cost. Instead, we may resort to balancing between reliability and cost, and seeking a good trade-off. Though some previous researchers did attempt to do so [12], they still solved the problem in a single-objective optimization manner, where the resource allocation scheme is optimized with respect to a weighted sum of reliability and testing cost. This might be inappropriate because reliability and cost are generally of different scales. Summing them up does not really provide meaningful information about the quality of solutions, and might cause difficulties in practice as it is hard to determine the appropriate values of weights.

In a preliminary work [25], we proposed employing Multi-Objective Evolutionary Algorithms (MOEAs) to solve OTRAPs. Specifically, we formulated OTRAPs as two types of multi-objective problems. First, we considered the reliability of the system, and the testing cost as two separate objectives. Second, the total testing resource consumed was taken into account as the third goal. The advantages of MOEAs over existing approaches were then evaluated by applying two MOEAs, namely Nondominated Sorting Genetic Algorithm II (NSGA-II) [26], and multi-objective differential evolution, on two simple parallel-series modular software systems. This paper substantially extends our previous work, and can be distinguished from it from three aspects as follows.

1) In [25], we found that NSGA-II performs well on the first problem formulation, but fails on the second one. Multi-objective differential evolution performs satisfactorily on both problem formulations with fine-tuned control parameters. However, multi-objective differential evolution involves too many control parameters, which are usually problem-dependent, and difficult to fine-tune. As a result, it is hard for a practitioner to utilize multi-objective differential evolution effectively. For this reason, we propose a novel MOEA called Harmonic Distance Based Multi-Objective Evolutionary Algorithm (HaD-MOEA) in this paper. HaD-MOEA not only performs well on both types of multi-objective formulations of OTRAPs, but also involves fewer control parameters than does multi-objective

differential evolution. Hence, it fills the gap between the two algorithms we adopted in the previous study.

2) In [25], the efficacy of MOEAs on OTRAPs was only verified on small-scale parallel-series modular software systems, which might not be sufficient to draw a more general conclusion. In this work, we have conducted extensive empirical studies on two additional software systems. One is a parallel-series modular software system with a larger scale than those investigated before. The other is a star-structure modular software system. In this way, we can evaluate the efficacy of MOEAs (in particular, HaD-MOEA) more comprehensively on different problem sizes, and system structures.
3) A sensitivity analysis of the proposed HaD-MOEA with respect to the model parameters (i.e., parameters of the software system) is presented. In addition, the computational cost of both NSGA-II, and HaD-MOEA are also investigated. These issues are indispensable for evaluating the utility of our approach in the real world, but were absent from the previous work.

The rest of this paper is organized as follows. Section II presents the problem formulations of two multi-objective OTRAPs based on two types of software systems. Section III introduces the related work on evolutionary multi-objective optimization, and proposes HaD-MOEA. In Section IV, NSGA-II and HaD-MOEA are experimentally compared with each other, and with some state-of-the-art single-objective approaches on four case studies including three parallel-series modular software systems, and a mixture of star-structure modular software systems. Finally, discussions, and conclusions are presented in Section V.

II. Problem Formulations

Because a software system typically consists of multiple modules, two issues need to be addressed in the problem formulation: the model that formulates the relationship between the testing resource and reliability (or cost) on a single module, and the structure of the system (i.e., the way that modules are organized to form the system). Following previous work, we address the former issue with SRGMs, which are defined based on the following four assumptions [13].

1) The process of fault removing can be described by a nonhomogeneous Poisson process (NHPP).
2) The software application is subject to failures at random times, caused by the remaining faults in the system.
3) Each time a failure occurs, the corresponding fault is immediately removed, and no new faults are introduced.
4) The mean number of faults detected in the time interval $(t, t+\Delta t)$ by the current testing resource expenditures is proportional to the mean number of remaining errors.

For the latter issue, two types of system structures are considered: a parallel-series structure, and a star structure.

A. Problem Formulations on Parallel-Series Modular Software Systems

Parallel-series modular software systems are the most popular simulated system models to describe real-world software systems [27]. They are comprised of groups of parallel modules, and serial modules. The basic structure of a parallel-series modular software system with n groups of parallel modules (the group i consists of k_i modules) and m serial modules is demonstrated in Fig. 1. Based on SRGMs, the reliability of a parallel-series modular software system, and its associated testing cost can be quantified using (1)(5).

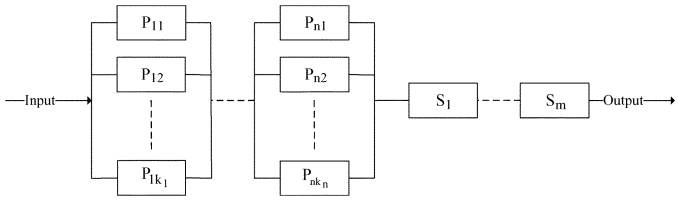

Fig. 1. Basic structure of a parallel-series modular software system.

The failure intensity of module i in a system is denoted as $\lambda_i(T_i)$, which can be calculated as

$$\lambda_i(T_i) = a_i b_i \exp\{-b_i T_i\}, \tag{1}$$

where a_i, and b_i are constants. a_i is the mean value of the total errors in module i, and b_i is the rate of detected errors in module i.

The reliability of module i is calculated by

$$R_i(x|T_i) = \exp\{-\lambda_i(T_i)x\}, \quad x \geq 0. \tag{2}$$

Based on the above two equations, the reliability of a parallel-series modular software system can be calculated with

$$R(x|T) = \prod_{l=1}^{n}\left(1 - \prod_{i=1}^{k_l}[1 - R_{li}(x|T_{li})]\right)\prod_{j=1}^{m} R_j(x|T_j), \tag{3}$$

where $(1 - \prod_{i=1}^{k_l}[1 - R_{li}(x|T_{li})])$ is the reliability of the lth parallel group. There are n groups of parallel modules, thus the total reliability of these n groups of parallel modules is $\prod_{l=1}^{n}(1 - \prod_{i=1}^{k_l}[1 - R_{li}(x|T_{li})])$. $\prod_{j=1}^{m} R_j(x|T_j)$ is the total reliability of the m serial modules.

The testing cost for each module is defined as [12]

$$C_i(R_i) = H_i \times \exp\{B_i R_i - D_i\}, \tag{4}$$

where H_i, B_i, and D_i are constants that control the increment speed of the testing cost corresponding to the reliability in module i. Correspondingly, the testing cost for the whole parallel-series modular software system is

$$C = \sum_{l=1}^{n}\sum_{i=1}^{ki_l} C_{li}(R_{li}) + \sum_{j=1}^{m} C_j(R_j), \tag{5}$$

where $\sum_{l=1}^{n}\sum_{i=1}^{k_l} C_{li}(R_{li})$ is the total testing cost for the n groups of parallel modules, and $\sum_{j=1}^{m} C_j(R_j)$ is the total testing cost for the m series modules.

Given an OTRAP, we aim at maximizing the reliability, while minimizing the testing cost. Meanwhile, the consumed testing resource should not exceed a pre-defined budget or (ideally) be minimized. However, (4) shows that the testing cost exponentially increases with reliability. Hence, the objectives of maximizing reliability and minimizing testing cost conflict with each

other. Such a scenario is not uncommon in the real world, and has been investigated under the name of multi-objective optimization. Mathematically, a multi-objective problem with m conflicting minimizing objectives can be formulated as (6).

$$\begin{aligned} \textit{Minimize} \quad & \mathbf{y} = f(x) = (f_1(x), f_2(x), \ldots, f_m(x)) \\ \textit{subject to} \quad & \mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbf{X} \\ & \mathbf{y} = (y_1, y_2, \ldots, y_m) \in \mathbf{Y}, \end{aligned} \tag{6}$$

where $\mathbf{x}$, $\mathbf{X}$, $\mathbf{y}$, and $\mathbf{Y}$ are called *decision vector*, *decision space*, *objective vector*, and *objective space*, respectively. $f_i(x)$ is the ith objective function of the problem. Because different objective functions usually conflict with each other, there seldom exists a unique solution that is optimal in terms of all objective functions. For this reason, the common approach to a multi-objective optimization problem is to seek a set of Pareto optimal solutions. In other words, each solution should not be inferior to any other solution on all objectives. Such solutions are referred to as nondominated solutions.

When formulating OTRAPs as multi-objective optimization problems, we first simultaneously consider reliability and cost as two objectives, and formulate OTRAPs as bi-objective problems. Despite the increase of the size and complexity of software systems, the cycle of a software development process has become shorter. Under such a circumstance, software testers might want to shorten the testing phase, even with the price of a slight decrease in reliability. So, we further consider the total testing resource expenditure as a new target, and a tri-objective problem can be formulated. Table I presents the mathematical formulation of the two multi-objective OTRAPs on parallel-series modular software systems.

TABLE I
FORMULATIONS OF TWO MULTI-OBJECTIVE PROBLEMS ON PARALLEL-SERIES MODULAR SOFTWARE SYSTEMS (MAXIMUM TESTING RESOURCE TO BE ALLOCATED IS T)

1. Bi-objective problem

$$\begin{cases} \text{Maximize} & R = \prod_{l=1}^{n}(1 - \prod_{i=1}^{k_l}[1 - R_{li}(x|T_{li})]) \prod_{j=1}^{m} R_j(x|T_j) \\ \text{Minimize} & C = \sum_{l=1}^{n}\sum_{i=1}^{k_l} C_{li}(R_{li}) + \sum_{j=1}^{m} C_j(R_j) \\ \text{s.t.} & \sum_{l=1}^{n}\sum_{i=1}^{k_l} T_{li} + \sum_{j=1}^{m} T_j \le T,\ T_{li}, T_j \ge 0 \end{cases}$$

2. Tri-objective problem

$$\begin{cases} \text{Maximize} & R = \prod_{l=1}^{n}(1 - \prod_{i=1}^{k_l}[1 - R_{li}(x|T_{li})]) \prod_{j=1}^{m} R_j(x|T_j) \\ \text{Minimize} & C = \sum_{l=1}^{n}\sum_{i=1}^{k_l} C_{li}(R_{li}) + \sum_{j=1}^{m} C_j(R_j) \\ \text{Minimize} & T_c = \sum_{l=1}^{n}\sum_{i=1}^{k_l} T_{li} + \sum_{j=1}^{m} T_j \\ \text{s.t.} & \sum_{l=1}^{n}\sum_{i=1}^{k_l} T_{li} + \sum_{j=1}^{m} T_j \le T,\ T_{li}, T_j \ge 0 \end{cases}$$

B. Problem Formulations on Star-Structure Modular Software Systems

Star-structure modular software systems are comprised of two types of units: central, and non-central. The central unit acts as a "server," and can be comprised of parallel, serial, or parallel-series modules. A non-central unit consists of some connected modules, each of which is an input of the whole system. Fig. 2 illustrates a star-structure modular software system with n non-central modules, and m central modules.

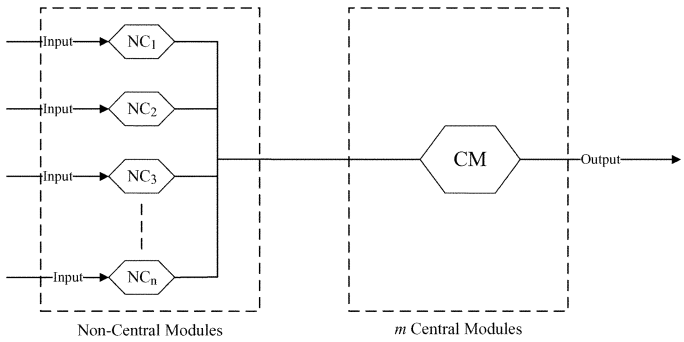

Fig. 2. Basic structure of a star-structure modular software system.

Equations (1), (2), and (4) can be readily utilized to quantify the reliability and cost of every single module in a star-structure modular software system. On the basis of these equations, the reliability of a star-structure modular software system can be calculated as

$$R(x|T) = \prod_{i=1}^{n} R_i^{\bar{c}}\left(x|T_i^{\bar{c}}\right) \times R_C(x|T_C)^n, \tag{7}$$

where $R_i^{\bar{c}}(x|T_i^{\bar{c}})$ is the reliability of the ith module in the non-central unit, and $R_C(x|T_C)$ is the reliability of the central unit (i.e., total reliability of the m central modules).

Similarly, the cost of testing a star-structure modular software system can be calculated by (8).

$$C = \sum_{i=1}^{n} C_i^{\bar{c}}\left(R_i^{\bar{c}}\right) + \sum_{j=1}^{m} C_j^{c}(R_j^{c}), \tag{8}$$

where $C_i^{\bar{c}}(R_i^{\bar{c}})$ is the cost of the ith module in the non-central unit, and $C_j^{c}(R_j^{c})$ stands for the cost of the jth module of the central unit. Similar to parallel-series modular software systems, the bi-objective, and tri-objective OTRAPs on star-structure modular software systems are mathematically formulated in Table II.

III. HARMONIC DISTANCE BASED MOEA FOR MULTI-OBJECTIVE OTRAPS

In the past few decades, evolutionary algorithms have emerged as an effective approach to multi-objective optimization problems [28], [29]. A lot of so called MOEAs, such as the Vector Evaluated Genetic Algorithm (VEGA) [30], the Pareto Archived Evolution Strategy (PAES) [31], and the Nondominated Sorting Genetic Algorithm II (NSGA-II), have been proposed. Proposed by Schaffer in 1985, VEGA is known as the first MOEA in literature. After that, the Nondominated Sorting

TABLE II
FORMULATIONS OF TWO MULTI-OBJECTIVE PROBLEMS ON STAR-STRUCTURE MODULAR SOFTWARE SYSTEMS (MAXIMUM TESTING RESOURCE TO BE ALLOCATED IS T)

1. Bi-objective problem

$$\begin{cases} \text{Maximize} & R = \prod_{i=1}^{n} R_i^{\bar{c}}(x|T_i^{\bar{c}}) \times R_C(x|T_C)^n \\ \text{Minimize} & C = \sum_{i=1}^{n} C_i^{\bar{c}}(R_i^{\bar{c}}) + \sum_{j=1}^{m} C_j^{c}(R_j^{c}) \\ \text{s.t.} & \sum_{i=1}^{n} T_i^{\bar{c}} + \sum_{j=1}^{m} T_j^{c} \le T,\ T_i^{\bar{c}}, T_j^{c} \ge 0 \end{cases}$$

2. Tri-objective problem

$$\begin{cases} \text{Maximize} & R = \prod_{i=1}^{n} R_i^{\bar{c}}(x|T_i^{\bar{c}}) \times R_C(x|T_C)^n \\ \text{Minimize} & C = \sum_{i=1}^{n} C_i^{\bar{c}}(R_i^{\bar{c}}) + \sum_{j=1}^{m} C_j^{c}(R_j^{c}) \\ \text{Minimize} & T_c = \sum_{i=1}^{n} T_i^{\bar{c}} + \sum_{j=1}^{m} T_j^{c} \\ \text{s.t.} & \sum_{i=1}^{n} T_i^{\bar{c}} + \sum_{j=1}^{m} T_j^{c} \le T,\ T_i^{\bar{c}}, T_j^{c} \ge 0 \end{cases}$$

Genetic Algorithm (NSGA) [32], Niched- Pareto Genetic Algorithm (NPGA) [33], and Multi-Objective Genetic Algorithm (MOGA) [34] were developed. Unlike VEGA, which adopts an objective-based fitness assignment strategy, these three methods assign fitness based on the concept of domination. Intuitively speaking, the latter three methods differ from VEGA in the way of selecting good solutions that are preserved in the problem-solving process. More recently, some other novel techniques have been developed for new MOEAs. For example, the niching technique has been suggested for preserving the diversity of the obtained solutions, based on which NSGA-II, and PAES were proposed [26], [31]. Elitism was also proven to be an effective factor for the convergence of an MOEA. Hence, most state-of-the-art MOEAs, including the Strength Pareto Evolutionary Algorithm (SPEA) [35], SPEA2 [36], PAES, and NSGA-II, were designed to preserve the nondominated solutions found during the problem-solving process.

Alg. 1. The Pseudo-Code of NSGA-II [26].

1: **Initialize:** Set the population size to N, and randomly generate the parent population $P_0 = \{x_1, x_2, \ldots, x_N\}$.

2: Set the generation number $t = 0$.

3: **while** $t < t_{max}$ (the terminate generation number) **do**

4: Generate the offspring population Q_t from P_t with the same population size.

5: Combine the parent and offspring population via $R_t = P_t \bigcup Q_t$.

6: Sort all solutions of R_t to get all non-dominated fronts $F = \text{fast}-\text{non}-\text{dominated}-\text{sort}(R_t)$ where $F = (F_1, F_2, \ldots)$.

7: Set $P_{t+1} = \phi$, $i = 1$.

8: **while** the parent population size $|P_{t+1}| + |F_i| < N$ **do**

9: (a) calculate crowding-distance of F_i.

10: (b) add the ith non-dominated front F_i to the parent pop P_{t+1}.

11: (c) $i = i + 1$.

12: **endwhile**

13: Sort the F_i according to the crowding distance.

14: Fill the parent pop P_{t+1} with the first $N - |P_{t+1}|$ elements of F_i.

15: Set $t = t + 1$.

16: **endwhile**

17: **return** P_t.

Among all the above-mentioned MOEAs, NSGA-II has shown superior performance on not only benchmark problems [37], [38], but also real-world application [39], and thus has been an off-the-shelf choice for solving multi-objective optimization problems. For this reason, we first employed NSGA-II to tackle the multi-objective OTRAPs on parallel-series modular software systems in a previous study [25]. However, empirical studies showed that NSGA-II managed to solve the bi-objective problem, but failed on the tri-objective problem. Although some recent studies have revealed that the performance of MOEAs may deteriorate when the number of objectives of a problem increases [40], it is still necessary to investigate why NSGA-II failed in the context of OTRAPs. Therefore, we have investigated this issue in-depth, and found that the failure of NSGA-II may be due to two reasons:

1) **The crowding distance measure does not accurately reflect the crowding degree**.
Alg. 1 presents the pseudo-code of NSGA-II. One critical procedure in NSGA-II is the calculation of crowding distance of a solution in the objective space [26]. Given a solution (a point in the objective space), the crowding distance is calculated as the 1-norm distance between the two nearest neighbors of the solution. In a multi-objective optimization problem, it is usually expected that the final solutions cover the whole objective space well. Hence, in the problem-solving process, the solutions with larger crowding distances are preferable. The first drawback of NSGA-II is that it employs a crowding distance metric which does not accurately reflect the actual crowing degree of a given solution. Fig. 3 illustrates such a scenario. We would say that solution C is more crowded than solution Y because C is quite close to D, and thus preserving one of them should be enough. However, in NSGA-II, the crowding distance of C is larger than that of Y. Hence,

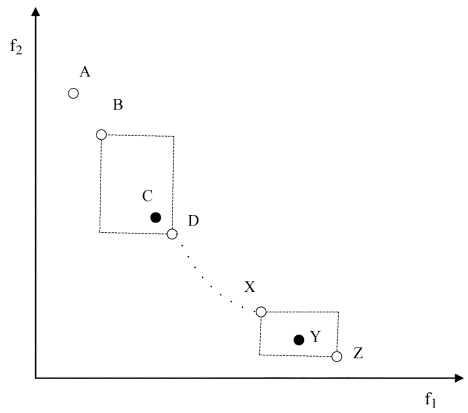

Fig. 3. Crowding degree estimation.

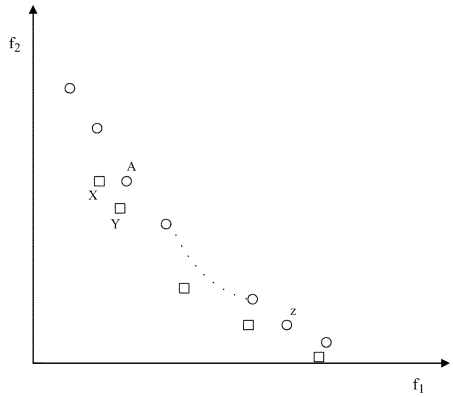

Fig. 4. Crowding degree estimation.

NSGA-II may obtain more solutions in one region of the objective space, while obtaining fewer solutions in some other regions.

2) **The selection strategy based on the crowding distance is unilateral.**
Because all MOEAs search for the solutions in an iterative way, the most important part of an MOEA is how it operates in a single iteration (usually referred to as a generation in the evolutionary computation literature). From Alg. 1, we can observe that, in each generation of NSGA-II, the parent population P_t and offspring population Q_t are first combined to make an intermediate population R_t. Then, R_t is sorted into different nondominated fronts through the function fast-non-dominated-sort; and for every nondominated front, the crowding distance of each individual in it is calculated. The parent population for the next generation is finally selected from R_t based on their nondominated levels and crowding distances. During this process, when calculating the crowding distance of a solution, only those solutions belonging to the same nondominated front are considered. However, this may lead to an inappropriate selection of solutions. For example, the rectangular, and circular points in Fig. 4 belong to two nondominated fronts. Assume that all of the rectangular points have been selected, and we need to select some circular points based on the crowding distance. Based on the selection strategy of NSGA-II, point A will be selected prior to point B, but this is somewhat counter-intuitive because point B is obviously less crowded than point A.

Having the above two drawbacks of NSGA-II in mind, we propose two alternative approaches to deal with them. First, instead of calculating the crowding distance using the 1-norm distance between the two nearest points to a given solution, we calculate it with the harmonic average distance. Assume the distances of the k-nearest solutions of a solution are $d_1, d_2, \ldots, d_k$. Then the harmonic average distance associated with this solution can be calculated as

$$d = \frac{k}{\frac{1}{d_1} + \frac{1}{d_2} + \cdots + \frac{1}{d_k}} \tag{9}$$

To overcome the second drawback of NSGA-II, we propose the following scheme. After sorting the solutions in the intermediate population into a number of fronts, if some solutions are to be selected from a front, the crowding distance will be calculated based on both the solutions belonging to the same front, and all the previously selected solutions. In this way, the undesirable scenario illustrated in Fig. 4 will be prevented.

Incorporating the above two schemes into the framework of NSGA-II, we obtain a new MOEA, the HaD-MOEA. The pseudo-code of HaD-MOEA is shown in Alg. 2. Specifically, lines 12-15 present the detailed steps of the two novel schemes adopted by HaD-MOEA. Taking advantage of the two new schemes, we expect the solutions obtained by HaD-MOEA to spread better in the objective space than those obtained by NSGA-II.

When applying HaD-MOEA to a system with n modules, a solution is encoded by an n-dimensional vector (chromosome). Each element represents the testing time consumed by a module, and the sum of these elements should not exceed the total testing time T. Besides, HaD-MOEA cannot guarantee always generating solutions that satisfy this constraint during search. Therefore, whenever a solution obtained violates the constraint, it will be repaired by using the procedures presented in Table III.

Alg. 2. The Pseudo-Code of HaD-MOEA.

1: **Initialize:** Set the population size to N, and randomly generate the parent population $P_0 = \{x_1, x_2, \ldots, x_N\}$.

2: Set the generation number $t = 0$.

3: **while** $t <$ the terminate generation number **do**

4: Generate the offspring population Q_t from P_t with the same population size.

5: Combine the parent and offspring population via $R_t = P_t \bigcup Q_t$.

6: Sort all solutions of R_t to get all non-dominated fronts $F = \text{fast}-\text{non}-\text{dominated}-\text{sort}(R_t)$ where $F = (F_1, F_2, \ldots)$.

7: Set $P_{t+1} = \phi$, and $i = 1$.

8: **while** the parent population size $|P_{t+1}| + |F_i| < N$ **do**

TABLE III
REPAIRING AN INFEASIBLE OFFSPRING $a' = (t'_1, t'_2, \ldots t'_n)$

Assume an infeasible solution a' is re-assigned to a new chromosome a''.

1: First we calculate the sum of elements in a':$S = sum(a')$.

2: For the bi-objective problem, the a'' is

$$a''=(t''_1, t''_2, \ldots t''_n) = ((t'_1 \times T/S), (t'_2 \times T/S), \ldots (t'_n \times T/S)).$$

3: For the tri-objective problem, the a'' is

$$a''=(t''_1, t''_2, \ldots t''_n) = ((t'_1 \times T \times r(0,1)/S), (t'_2 \times T \times r(0,1)/S), \ldots (t'_n \times T \times r(0,1)/S)),$$

$r(0,1)$ is a random number between 0 and 1.

9: add the ith non-dominated front F_i to the parent pop P_{t+1}.

10: $i = i + 1$.

11: **endwhile**

12: Combine F_i and P_{t+1} to a temporary vector T.

13: Calculate the harmonic crowding distances of individuals of F_i in T.

14: Sort the F_i according to the crowding distance.

15: Set $T = \phi$.

16: Fill the parent pop P_{t+1} with the first $N - |P_{t+1}|$ elements of F_i.

17: Set $t = t + 1$.

18: **endwhile**

19: **return** P_t.

IV. EXPERIMENTAL STUDIES

Experimental studies have been carried out to evaluate the efficacy of our proposed approach. The experimental studies were designed to consist of two parts. First, the MOEAs were compared with three state-of-the-art single-objective approaches to evaluate whether they (in particular HaD-MOEA) are able to provide some advantage over the single-objective approaches. Second, HaD-MOEA was compared with NSGA-II in terms of both solution quality, and computational time.

A. Comparing MOEAs With Single-Objective Approaches

Three single-objective approaches were considered in this experiment. For brevity, these approaches will be denoted using the acronyms of the family names of the researchers who proposed them. The first algorithm, denoted as the D-X algorithm, was proposed by Dai and Xie *et al.* [12]. It is essentially a genetic algorithm that aims to solve a single-objective problem whose objective function is the weighted sum of the reliability and cost. The second algorithm was proposed by Yang and Xie [9], and thus is denoted as the Y-X algorithm. It considers only the software reliability, and ignores the cost. The last method, denoted as the T-M algorithm, was devised by Tom and Murthy [41], who considered the problem of finding an allocation of program modules onto processors of a distributed computing system with the single goal to maximize the reliability while ignoring the cost.

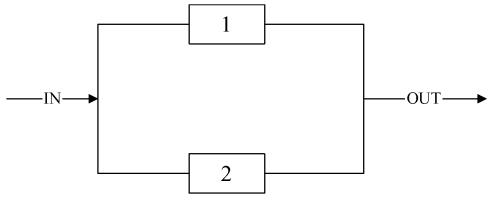

Fig. 5. Structure of the simple parallel-series modular software system with two modules.

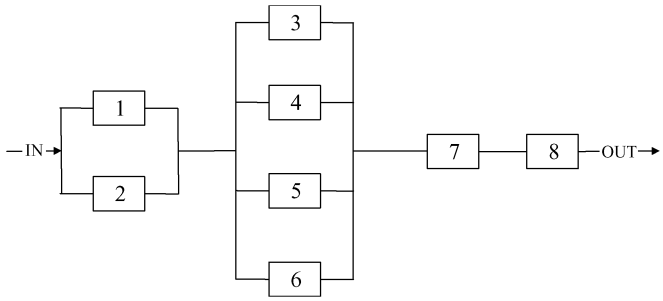

Fig. 6. Structure of the complex parallel-series modular software system with eight modules.

We compared the MOEAs with the single-objective approaches on two parallel-series modular software systems: A simple parallel-series modular software system with only two parallel modules, and a complex parallel-series modular software system with eight modules. The structures of the two systems are illustrated in Figs. 5 and 6. The Y-X, and T-M algorithms have been separately applied to the simple, and complex parallel-series modular software systems. The D-X algorithm has been applied to both. Hence, the two systems serve as a good platform for our comparative study. It is noteworthy that all the compared single-objective algorithms do not consider the total testing time as an objective. Hence, we only consider the bi-objective problem on the two systems in this experiment. All results of the single-objective approaches were obtained from the original publications.

Following the previous studies, we assume that 10 persons are available for testing the simple system, and 23 persons are available for the complex system. Each person can spend up to 1000 hours on the testing task. In addition, all the parameters of the systems are set to the values used in previous studies [9], [12], [41]. We ran the two MOEAs (NSGA-II, and HaD-MOEA) on each system for 30 independent runs. Similar results were obtained in the 30 runs. The solutions obtained in the first run on

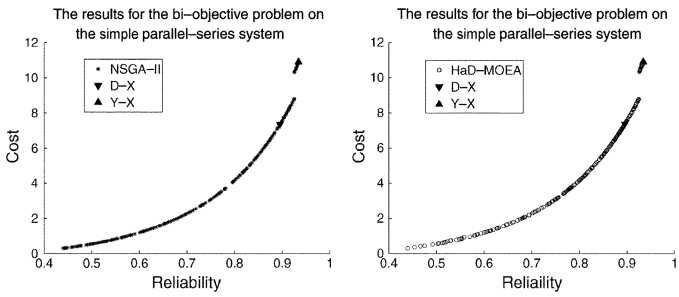

Fig. 7. Results for the bi-objective problem on the simple parallel-series modular software system.

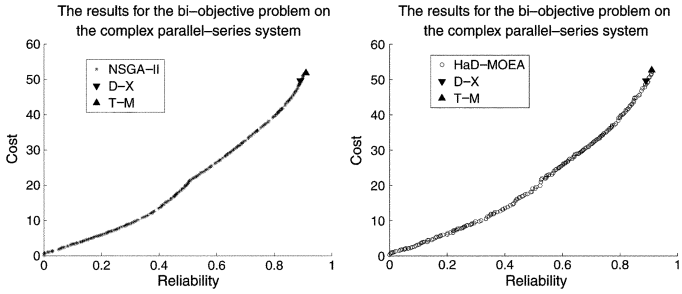

Fig. 8. Results for the bi-objective problem on the complex parallel-series modular software system.

TABLE IV
NEAREST SOLUTIONS OBTAINED BY MOEAS TO THE SOLUTIONS OBTAINED BY SINGLE-OBJECTIVE ALGORITHMS ON THE SIMPLE, AND COMPLEX PARALLEL-SERIES MODULAR SOFTWARE SYSTEMS

cost factor	simple system			complex system		
	Algorithm	Reliability	Cost(units)	Algorithm	Reliability	Cost(units)
Consider	NSGA-II	0.8953	7.356	NSGA-II	0.8957	49.46
Consider	HaD-MOEA	0.8953	7.355	HaD-MOEA	0.8927	49.04
consider	D-X	0.8953	7.356	D-X	0.89	49.65
Consider	NSGA-II	0.9344	10.903	NSGA-II	0.9055	51.88
Consider	HaD-MOEA	0.9344	10.903	HaD-MOEA	0.9104	52.33
Without	Y-X	0.9344	10.903	T-M	0.91	52.64

the simple system are illustrated in Fig. 7, together with the solutions obtained by D-X, and Y-X algorithms. In addition, the solutions obtained in the first run on the complex system are illustrated in Fig. 8, together with the solutions obtained by D-X, and T-M algorithms. From the figures, it can be observed that both NSGA-II, and HaD-MOEA achieved such a set of solutions that provide a variety of trade-offs between the system reliability and the cost. Furthermore, it would be interesting to check whether the solutions achieved by the single-objective approaches can also be obtained by MOEAs. For this reason, we recorded all nondominated solutions obtained by the MOEAs in the 30 runs. Then, the difference between the reliability of these solutions and the solutions obtained by the single-objective approaches were calculated. The nondominated solutions corresponding to the smallest difference were identified, and presented in Table IV.

The MOEAs managed to find the same or very similar solutions as that provided by the single-objective approaches.

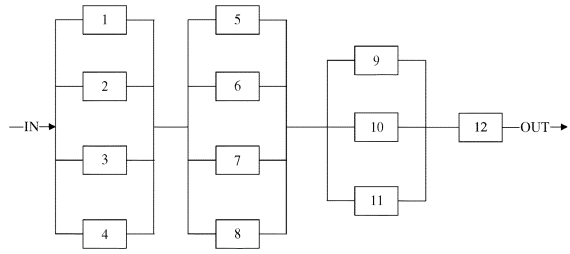

Fig. 9. Structure of the larger parallel-series modular software system with twelve modules.

However, MOEAs find many additional solutions with different trade-offs between the reliability and cost, which enable a software tester to ask what-if questions. Such a set of different solutions also enables a software tester to see the exact relationship between the reliability and cost. It is now possible to ask practical questions, such as how much savings one can obtain if we were willing to sacrifice the reliability by a certain amount. Therefore, compared to the single-objective approaches, the major advantage of using MOEAs on OTRAP is that it can provide a lot of additional choices to a practitioner, and thus benefit organizing the whole testing process.

B. Comparing HaD-MOEA With NSGA-II

According to the results presented in the above subsection, we can find that NSGA-II and HaD-MOEA performed comparably for the bi-objective problem on the simple, and complex parallel-series modular software systems. Hence, a natural question is whether the two algorithms perform comparably on all OTRAPs? In other words, is NSGA-II alone sufficient for OTRAPs? To answer this question, we have further conducted comprehensive experiments to compare HaD-MOEA with NSGA-II.

In this experiment, we considered both types of multi-objective problems that we formulated in Section II. The simple, and complex parallel-series modular software systems were utilized as the test-bed. In addition, the efficacy of the MOEAs on another two systems was also evaluated. The first system is the parallel-series modular software system illustrated in Fig. 9. It consists of twelve modules. The purpose of evaluating the MOEAs on this system is to verify the MOEAs' scalability with respect to the number of modules. As shown in Fig. 10, the second system is a star-structure modular software system (denoted as the SS-system) with six modules. By applying the MOEAs to this system, we aim to check whether the MOEAs can work well on different types of structures.

For the parallel-series modular software systems, we assumed the same total testing time as in the previous sub-section. We assumed that 35, and 15 persons are available for the larger parallel-series modular software system, and the star-structure modular software system. The testing time for each person was set to 1000 hours. In the previous sub-section, all algorithms were evaluated using a set of pre-defined parameters of the systems. It is necessary to evaluate the sensitivity of the MOEAs to these system parameters. Therefore, we further randomly generated 30 different parameter settings for each system. For each

TABLE V
RANGE OF THE PARAMETER VALUES OF THE FOUR SOFTWARE SYSTEMS (THE VALUE OF THE PARAMETER IS A REAL NUMBER WHICH IS RANDOMLY GENERATED WITHIN THE CORRESPONDING RANGE). a_i,b_i, H_i, B_i, AND D_i ARE PARAMETERS OF THE iTH MODULE IN SOFTWARE SYSTEMS

	a_i		b_i		H_i	B_i	D_i	
Simple system	$i = 1:2$		$i = 1:2$		$i = 1:2$	$i = 1:2$	$i = 1:2$	
	$[150.0, 250.0]$		[5.5E-4, 6.0E-4]		$[4.5, 5.5]$	$[5.5, 7.0]$	$[4.5, 5.5]$	
Complex system	$i = 1:6$	$i = 7:8$	$i = 1:6$	$i = 7:8$	$i = 1:8$	$i = 1:8$	$i = 1:6$	$i = 7:8$
	$[200.0, 350.0]$	$[30.0, 35.0]$	[3.0E-4, 9.0E-4]	[5.8E-3, 6.2E-3]	$[3.4, 3.55]$	$[6.0, 6.2]$	$[4.9, 5.1]$	$[4.0, 4.1]$
Larger system	$i = 1:11$	$i = 12$	$i = 1:11$	$i = 12$	$i = 1:12$	$i = 1:12$	$i = 1:11$	$i = 12$
	$[200.0, 350.0]$	$[30.0, 35.0]$	[3.0E-4, 9.0E-4]	[5.8E-3, 6.2E-3]	$[3.4, 3.55]$	$[6.0, 6.2]$	$[4.9, 5.1]$	$[4.0, 4.1]$
SS-system	$i = 1:5$	$i = 6$	$i = 1:5$	$i = 6$	$i = 1:6$	$i = 1:6$	$i = 1:5$	$i = 6$
	$[200.0, 350.0]$	$[30.0, 35.0]$	[3.0E-4, 9.0E-4]	[5.8E-3, 6.2E-3]	$[3.4, 3.55]$	$[6.0, 6.2]$	$[4.9, 5.1]$	$[4.0, 4.1]$

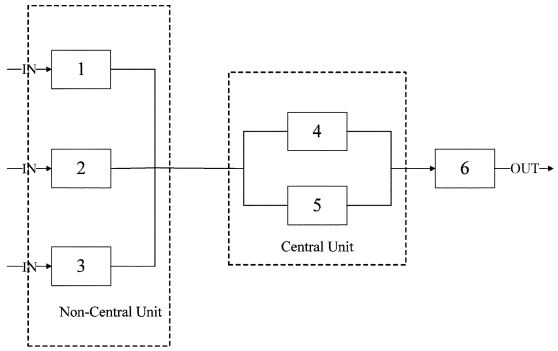

Fig. 10. Structure of the SS-system with six modules.

TABLE VI
PARAMETER SETTINGS OF NSGA-II, AND HAD-MOEA

software systems	problem type	crossover probability	mutation probability	terminate generation	population size
simple system	bi-objective problem	0.9	0.1	200	200
simple system	tri-objective problem	0.9	0.1	200	500
complex system	bi-objective problem	0.9	0.1	200	200
complex system	tri-objective problem	0.9	0.1	200	500
larger system	bi-objective problem	0.9	0.1	200	200
larger system	tri-objective problem	0.9	0.1	200	500
SS-system	bi-objective problem	0.9	0.1	200	200
SS-system	tri-objective problem	0.9	0.1	200	500

module of each system, all 30 parameter settings were generated within a pre-defined interval, as presented in Table V. Given a system with a randomly generated parameter setting, both HaD-MOEA, and NSGA-II have been applied to it for 30 independent runs. To make the comparison as fair as possible, the two algorithms adopted exactly the same initial populations and control parameters in each run. Table VI presents the control parameters used throughout the experiment. To summarize, we altogether compared the two MOEAs on 240 (two types of problems $\times$ four systems $\times$ 30 system parameter settings) problem instances, and carried out a statistical test on each instance based on the results of the 30 independent runs.

TABLE VII
VALUES OF HYPERVOLUME INDICATOR OF OBTAINED RESULTS ON DIFFERENT PARAMETER SUITES ON THE SIMPLE PARALLEL-SERIES MODULAR SOFTWARE SYSTEM (THE RESULT THAT IS SIGNIFICANTLY BETTER THAN THE OTHER IS EMPHASIZED IN **BOLDFACE**)

parameter suite	bi-objective problem						tri-objective problem					
	NSGA-II			HaD-MOEA			NSGA-II			HaD-MOEA		
	best	worst	mean	best	worst	mean	best	worst	mean	best	worst	mean
No.1	0.8330	0.8328	0.8329	0.8450	0.8498	**0.8499**	0.3633	0.3597	0.3621	0.3668	0.3651	**0.3662**
No.2	0.8018	0.7858	0.7866	0.7908	0.7906	0.7907	0.3485	0.3450	0.3473	0.3519	0.3505	**0.3514**
No.3	0.8631	0.8603	0.8605	0.8704	0.8680	**0.8683**	0.3734	0.3686	0.3719	0.3775	0.3723	0.3749
No.4	0.8838	0.8832	**0.8833**	0.8593	0.8591	0.8592	0.3824	0.3796	0.3813	0.3842	0.3813	0.3835
No.5	0.8844	0.8802	**0.8804**	0.8712	0.8712	0.8712	0.3428	0.3323	**0.3376**	0.3363	0.3320	0.3337
No.6	0.8691	0.8005	0.8029	0.8615	0.8609	0.8610	0.3742	0.3724	0.3733	0.3765	0.3751	**0.3758**
No.7	0.8479	0.8477	0.8478	0.8471	0.8360	0.8365	0.3266	0.3214	0.3252	0.3298	0.3290	**0.3294**
No.8	0.8115	0.8113	**0.8114**	0.7850	0.7849	0.7849	0.3717	0.3667	0.3695	0.3736	0.3721	**0.3728**
No.9	0.7986	0.7947	**0.7949**	0.7758	0.7702	0.7704	0.3222	0.3211	0.3218	0.3238	0.3210	0.3224
No.10	0.8494	0.8491	**0.8492**	0.8289	0.8288	0.8289	0.3499	0.3437	0.3476	0.3538	0.3499	**0.3513**
No.11	0.8562	0.8553	0.8553	0.8644	0.8640	**0.8641**	0.3568	0.3549	0.3555	0.3590	0.3572	**0.3578**
No.12	0.8866	0.8865	**0.8865**	0.8595	0.8583	0.8585	0.3993	0.3946	0.3977	0.4025	0.4012	**0.4020**
No.13	0.8476	0.8467	**0.8470**	0.8289	0.8288	0.8289	0.3590	0.3589	**0.3590**	0.3588	0.3575	0.3584
No.14	0.7925	0.7559	0.7572	0.7840	0.7839	0.7840	0.3880	0.3854	0.3872	0.3916	0.3901	**0.3908**
No.15	0.8665	0.8663	0.8665	0.8680	0.8661	0.8667	0.3225	0.3210	0.3218	0.3271	0.3217	0.3250
No.16	0.8213	0.8189	0.8193	0.8373	0.8372	**0.8372**	0.3368	0.3307	0.3342	0.3433	0.3420	**0.3425**
No.17	0.8796	0.8790	0.8793	0.8925	0.8874	**0.8882**	0.3423	0.3387	0.3409	0.3460	0.3448	**0.3455**
No.18	0.8535	0.8469	0.8472	0.8519	0.8518	0.8518	0.3510	0.3497	0.3501	0.3568	0.3515	**0.3527**
No.19	0.8283	0.8278	0.8280	0.8376	0.8376	**0.8376**	0.3797	0.3765	0.3789	0.3834	0.3826	**0.3831**
No.20	0.7691	0.7556	0.7563	0.7769	0.7769	**0.7769**	0.3648	0.3630	0.3643	0.3683	0.3677	**0.3681**
No.21	0.8496	0.8494	**0.8495**	0.8174	0.8173	0.8174	0.3291	0.3249	0.3271	0.3313	0.3293	**0.3303**
No.22	0.8172	0.8085	0.8089	0.8118	0.8117	0.8117	0.3336	0.3321	0.3327	0.3382	0.3266	0.3323
No.23	0.8586	0.8574	0.8575	0.8698	0.8690	**0.8691**	0.3733	0.3692	0.3721	0.3764	0.3750	**0.3758**
No.24	0.8615	0.8614	0.8614	0.8788	0.8785	**0.8785**	0.3479	0.3440	0.3467	0.3512	0.3450	**0.3507**
No.25	0.7997	0.7934	0.7937	0.8155	0.8154	**0.8155**	0.3738	0.3698	0.3727	0.3770	0.3758	**0.3764**
No.26	0.8649	0.8647	0.8648	0.8626	0.8625	0.8625	0.3832	0.3811	0.3827	0.3858	0.3843	**0.3851**
No.27	0.8559	0.8551	**0.8552**	0.8458	0.8426	0.8430	0.3211	0.3190	0.3202	0.3251	0.3237	**0.3245**
No.28	0.8539	0.8176	0.8190	0.8419	0.8417	0.8418	0.3575	0.3417	0.3431	0.3476	0.3406	0.3424
No.29	0.8273	0.8269	**0.8271**	0.8141	0.8140	0.8141	0.3105	0.3066	0.3096	0.3132	0.3117	**0.3127**
No.30	0.8728	0.8722	**0.8723**	0.8587	0.8584	0.8584	0.3422	0.3414	0.3417	0.3479	0.3437	**0.3445**

Fig. 11 illustrates the nondominated solutions obtained on the tri-objective problems in the first run of the first randomly generated parameter settings. The solutions obtained by HaD-MOEA spread better in the objective space than those obtained by NSGA-II. This result suggests that HaD-MOEA might be advantageous to NSGA-II. However, the figures are just a simple illustration, and no rigorous conclusion can be made yet.

TABLE VIII
VALUES OF HYPERVOLUME INDICATOR OF OBTAINED RESULTS ON DIFFERENT PARAMETER SUITES ON THE COMPLEX PARALLEL-SERIES MODULAR SOFTWARE SYSTEM (THE RESULT THAT IS SIGNIFICANTLY BETTER THAN THE OTHER IS EMPHASIZED IN **BOLDFACE**)

parameter suite	bi-objective problem						tri-objective problem					
	NSGA-II			HaD-MOEA			NSGA-II			HaD-MOEA		
	best	worst	mean	best	worst	mean	best	worst	mean	best	worst	mean
No.1	0.5581	0.5122	0.5459	0.5694	0.5559	0.5637	0.2281	0.1440	0.2065	0.2221	0.2113	0.2140
No.2	0.5519	0.5183	0.5403	0.5444	0.5245	0.5359	0.1969	0.1685	0.1833	0.1998	0.1935	**0.1956**
No.3	0.5630	0.5074	0.5470	0.5678	0.5431	**0.5591**	0.1917	0.0984	0.1497	0.1834	0.1385	0.1600
No.4	0.5547	0.5239	**0.5420**	0.5393	0.5256	0.5319	0.2231	0.1861	0.2160	0.2310	0.2275	**0.2295**
No.5	0.5301	0.4064	0.5121	0.5222	0.4927	0.5134	0.1629	0.0816	0.1309	0.1705	0.1241	**0.1521**
No.6	0.5666	0.5446	0.5562	0.5616	0.5489	0.5560	0.2278	0.1246	0.2014	0.2279	0.1992	0.2151
No.7	0.5482	0.5108	0.5301	0.5477	0.5311	0.5394	0.1790	0.1247	0.1622	0.1894	0.1781	**0.1864**
No.8	0.5537	0.5227	**0.5443**	0.5356	0.5204	0.5293	0.2251	0.1137	0.2106	0.2329	0.2190	0.2296
No.9	0.5248	0.4723	0.5066	0.5098	0.4861	0.4997	0.1443	0.0703	0.1240	0.1541	0.1420	**0.1510**
No.10	0.5618	0.4928	0.5503	0.5483	0.5308	0.5422	0.2197	0.1848	0.2088	0.2242	0.2109	0.2182
No.11	0.5364	0.4999	0.5192	0.5416	0.5249	**0.5348**	0.1858	0.1289	**0.1610**	0.1966	0.1365	0.1486
No.12	0.5471	0.5043	0.5295	0.5304	0.5175	0.5247	0.2125	0.1361	0.1943	0.2365	0.2132	**0.2201**
No.13	0.5569	0.4743	0.5426	0.5446	0.5314	0.5385	0.2245	0.1601	0.2068	0.2578	0.1833	0.2108
No.14	0.5716	0.5457	0.5602	0.5654	0.5467	0.5570	0.2362	0.1947	0.2139	0.2484	0.1979	0.2194
No.15	0.5380	0.4484	0.5198	0.5389	0.5145	**0.5284**	0.1773	0.1079	0.1577	0.1924	0.1489	**0.1623**
No.16	0.5669	0.5354	0.5531	0.5780	0.5653	**0.5723**	0.2391	0.1407	0.2262	0.2382	0.2305	**0.2364**
No.17	0.5261	0.4671	0.5094	0.5339	0.5016	0.5202	0.1505	0.0541	0.1023	0.1943	0.1183	**0.1389**
No.18	0.5723	0.5473	0.5600	0.5713	0.5536	0.5618	0.2189	0.1133	0.1771	0.2178	0.1402	0.1850
No.19	0.5206	0.4889	0.5123	0.5264	0.4671	0.5044	0.1369	0.0415	0.1218	0.1666	0.1221	**0.1403**
No.20	0.5530	0.5204	0.5371	0.5586	0.5387	**0.5501**	0.2105	0.1924	**0.2067**	0.2208	0.1844	0.1984
No.21	0.5589	0.5285	**0.5430**	0.5377	0.5207	0.5300	0.2111	0.1449	0.1937	0.2342	0.1532	**0.1977**
No.22	0.5110	0.4274	0.4967	0.5076	0.4361	0.4842	0.1257	0.0766	0.1117	0.1321	0.0944	0.1242
No.23	0.5533	0.5127	0.5413	0.5605	0.5453	0.5525	0.2255	0.1165	0.2100	0.2599	0.1802	**0.2366**
No.24	0.5104	0.4226	0.4925	0.5206	0.4728	0.5037	0.1343	0.0389	0.1168	0.1665	0.1198	**0.1444**
No.25	0.5569	0.5382	0.5498	0.5680	0.5528	**0.5609**	0.2144	0.1809	0.2065	0.2495	0.2000	0.2290
No.26	0.5489	0.5162	0.5341	0.5474	0.5175	0.5298	0.1603	0.1391	0.1489	0.1795	0.1301	0.1473
No.27	0.5538	0.5025	0.5377	0.5473	0.5251	0.5409	0.2048	0.1830	0.1983	0.2199	0.1924	0.2077
No.28	0.5437	0.5188	0.5318	0.5360	0.5108	0.5227	0.1808	0.0964	**0.1618**	0.1967	0.1183	0.1432
No.29	0.5387	0.4656	0.5210	0.5301	0.5004	**0.5174**	0.1589	0.0831	0.1391	0.1633	0.1035	0.1386
No.30	0.5184	0.4782	0.5030	0.5101	0.4566	0.4787	0.1250	0.0810	0.1014	0.1463	0.1104	**0.1363**

TABLE IX
VALUES OF HYPERVOLUME INDICATOR OF OBTAINED RESULTS ON DIFFERENT PARAMETER SUITES ON THE LARGER PARALLEL-SERIES MODULAR SOFTWARE SYSTEM (THE RESULT THAT IS SIGNIFICANTLY BETTER THAN THE OTHER IS EMPHASIZED IN **BOLDFACE**)

parameter suite	bi-objective problem						tri-objective problem					
	NSGA-II			HaD-MOEA			NSGA-II			HaD-MOEA		
	best	worst	mean	best	worst	mean	best	worst	mean	best	worst	mean
No.1	0.7916	0.7426	0.7747	0.8077	0.7712	0.7892	0.2770	0.2509	0.2662	0.2842	0.2553	**0.2717**
No.2	0.7949	0.7422	0.7771	0.7841	0.7377	0.7664	0.2835	0.2563	0.2700	0.2889	0.2626	**0.2768**
No.3	0.7813	0.7402	0.7638	0.7879	0.7354	0.7678	0.2618	0.2280	0.2454	0.2650	0.2296	0.2509
No.4	0.7617	0.6860	**0.7326**	0.7406	0.6796	0.7083	0.2458	0.2119	0.2326	0.2512	0.2293	**0.2480**
No.5	0.7580	0.5880	0.7293	0.7468	0.6751	**0.7128**	0.2241	0.1779	0.2074	0.2282	0.1851	**0.2124**
No.6	0.7361	0.4987	0.6692	0.7297	0.6696	**0.7030**	0.2033	0.1490	0.1854	0.2058	0.1828	**0.1999**
No.7	0.7649	0.6825	0.7362	0.7641	0.7125	0.7431	0.2279	0.1995	0.2175	0.2338	0.2050	0.2219
No.8	0.8045	0.7597	0.7875	0.7782	0.7418	0.7642	0.3235	0.2913	0.3054	0.3257	0.3016	**0.3126**
No.9	0.7744	0.6341	0.7482	0.7523	0.6925	0.7288	0.2422	0.2232	0.2318	0.2461	0.2350	**0.2394**
No.10	0.7723	0.6659	**0.7460**	0.7536	0.6882	0.7249	0.2340	0.1865	0.2120	0.2379	0.1904	0.2168
No.11	0.7972	0.7373	0.7792	0.8049	0.7664	**0.7902**	0.3044	0.2775	0.2916	0.3291	0.3055	**0.3183**
No.12	0.7822	0.66639	0.7481	0.7583	0.7046	0.7369	0.2637	0.2181	0.2482	0.2666	0.2268	**0.2538**
No.13	0.7785	0.6773	0.7497	0.7612	0.7244	0.7450	0.2530	0.2125	0.2306	0.2555	0.218	0.2356
No.14	0.7840	0.6510	0.7585	0.7756	0.7030	0.7559	0.2598	0.2382	0.2483	0.2633	0.2431	**0.2535**
No.15	0.7632	0.7178	**0.7322**	0.7646	0.6823	0.7282	0.2311	0.1940	0.2186	0.2363	0.1959	**0.2237**
No.16	0.7789	0.7163	0.7493	0.7941	0.7026	**0.7607**	0.2484	0.2088	0.2297	0.2502	0.2124	**0.2353**
No.17	0.7826	0.7229	0.7653	0.7941	0.7417	0.7683	0.2547	0.2110	0.2385	0.2633	0.2160	**0.2438**
No.18	0.7045	0.5542	0.6566	0.7032	0.5869	0.6395	0.1496	0.1171	0.1394	0.1534	0.1104	0.1425
No.19	0.7763	0.6278	0.7498	0.7851	0.7221	0.7541	0.2300	0.2042	0.2187	0.2394	0.2092	**0.2243**
No.20	0.8021	0.7652	0.7855	0.8103	0.7689	**0.7932**	0.2934	0.2526	0.2739	0.2960	0.2552	0.2795
No.21	0.7660	0.6966	**0.7439**	0.7370	0.6743	0.7019	0.2424	0.2019	0.2200	0.2507	0.2068	**0.2248**
No.22	0.7695	0.6816	0.7420	0.7644	0.6812	0.7310	0.2232	0.1870	0.2089	0.2290	0.1944	**0.2140**
No.23	0.7668	0.6897	0.7476	0.7768	0.7096	0.7459	0.2333	0.1893	0.2153	0.2426	0.1965	**0.2210**
No.24	0.7578	0.6556	0.7180	0.7730	0.6918	0.7368	0.2092	0.1632	0.1853	0.2104	0.1651	**0.1895**
No.25	0.7850	0.7295	0.7648	0.8006	0.7559	0.7821	0.2751	0.2408	0.2599	0.2769	0.2487	**0.2651**
No.26	0.7968	0.7399	0.7720	0.7947	0.7579	0.7788	0.2694	0.2340	0.2521	0.2747	0.2426	0.2584
No.27	0.7877	0.7342	0.7645	0.7785	0.7273	0.7667	0.2739	0.2079	0.2445	0.2761	0.2232	**0.2698**
No.28	0.7945	0.7541	0.7796	0.7834	0.7396	0.7697	0.2956	0.2624	0.2845	0.3057	0.2696	**0.2922**
No.29	0.7762	0.6759	0.7416	0.7639	0.7206	**0.7559**	0.2548	0.2101	0.2329	0.2566	0.2148	0.2382
No.30	0.7870	0.6804	0.7528	0.7743	0.7370	0.7550	0.2540	0.2267	0.2419	0.2599	0.2343	**0.2477**

To compare the MOEAs in a quantitative, rigorous way, a quantitative measure of the performance of the two algorithms is required. In the multi-objective optimization field, there are a number of metrics for measuring the goodness of the solutions obtained by an algorithm. Among these metrics, we adopted the hypervolume indicator in our study because it holds two important properties:

1) It is sensitive to any type of improvements, i.e., whenever an approximation set A dominates another approximation set B, the measure yields a strictly better quality value for the former than for the latter set [42].
2) As a result of the first property, the hypervolume measure guarantees that any approximation set A that achieves the maximally possible quality value for a particular problem contains all the Pareto-optimal objective vectors [43].

So far, the hypervolume indicator is the only metric that possesses the above two properties. Hence, it serves as the most appropriate metric for comparing NSGA-II and HaD-MOEA. For space considerations, we refrain from providing the detailed information of hypervolume indicator, but direct interested readers to [42].

Tables VII –X present the results (in terms of the values of hypervolume indicator) obtained by the MOEAs algorithms in 30 independent runs. For each algorithm, the best, median, and worst results are given. Two-sided Wilcoxon rank sum tests with a significance level 0.05 have been conducted to compare the performance of NSGA-II with that of HaD-MOEA. Table XI summarizes the results of the Wilcoxon test. When considering the bi-objective problems, HaD-MOEA performed comparably with NSGA-II on all four systems. This result is consistent with the results presented in the previous subsection. However, in case of the tri-objective problems, HaD-MOEA is obviously superior to NSGA-II. For all 120 testing instances, HaD-MOEA performed significantly better than NSGA-II on 73 instances, yet was inferior on only seven. To investigate the computational efficiency of the MOEAs, we recorded their runtime on all test instances. Table XII presents the average runtime for each problem on the four systems. In comparison with NSGA-II, HaD-MOEA merely involved an additional computational overhead of 0.4 seconds in the worst case, which was about 2% of the overall runtime of the whole algorithm. In essence, the computational efficiency of the two algorithms is comparable. Moreover, we may find that the runtime of both MOEAs are about 20 seconds for the bi-objective problems, and 130 seconds for the tri-objective problems, which are negligible compared to the total time that we allocated to the testing phase. This further justifies the utility of MOEAs in practice.

TABLE X
VALUES OF HYPERVOLUME INDICATOR OF OBTAINED RESULTS ON DIFFERENT PARAMETER SUITES ON THE SS-SYSTEM (THE RESULT THAT IS SIGNIFICANTLY BETTER THAN THE OTHER IS EMPHASIZED IN **BOLDFACE**)

parameter suite	bi-objective problem						tri-objective problem					
	NSGA-II			HaD-MOEA			NSGA-II			HaD-MOEA		
	best	worst	mean	best	worst	mean	best	worst	mean	best	worst	mean
No.1	0.4937	0.4873	0.4927	0.5037	0.4647	0.4851	0.3307	0.3274	0.3293	0.3308	0.3222	0.3301
No.2	0.4877	0.4856	**0.4865**	0.4811	0.4579	0.4700	0.3322	0.3292	0.3308	0.3423	0.3322	**0.3367**
No.3	0.4918	0.4889	0.4901	0.4959	0.4720	0.4859	0.3332	0.3305	0.3317	0.3362	0.3216	0.3291
No.4	0.4704	0.4452	**0.4636**	0.4574	0.4368	0.4457	0.3376	0.3312	0.3343	0.3585	0.3491	**0.3530**
No.5	0.5111	0.5085	0.5094	0.5035	0.4817	0.4950	0.3441	0.3404	0.3422	0.3449	0.3356	0.3407
No.6	0.4816	0.4583	0.4791	0.4774	0.4553	0.4678	0.3353	0.3319	0.3338	0.3453	0.3356	**0.3403**
No.7	0.4977	0.4948	**0.4963**	0.4972	0.4742	0.4889	0.3342	0.3314	**0.3332**	0.3336	0.3240	0.3297
No.8	0.4756	0.4693	0.4748	0.4600	0.4358	0.4506	0.3229	0.3210	0.3220	0.3309	0.3242	**0.3278**
No.9	0.4835	0.4587	0.4761	0.4697	0.4452	0.4572	0.3456	0.3432	0.3445	0.3450	0.3375	0.3418
No.10	0.4640	0.4442	0.4587	0.4528	0.4331	0.4434	0.3353	0.3311	0.3328	0.3343	0.3245	0.3294
No.11	0.4783	0.4759	0.4773	0.4829	0.4615	**0.4728**	0.3261	0.3228	0.3248	0.3283	0.3153	**0.3226**
No.12	0.5050	0.4650	**0.4949**	0.4896	0.4699	0.4791	0.3501	0.3465	0.3485	0.3489	0.3395	0.3460
No.13	0.4729	0.4504	0.4668	0.4625	0.4365	0.4504	0.3340	0.3312	0.3326	0.3432	0.3340	**0.3393**
No.14	0.4810	0.4203	0.4664	0.4758	0.4600	**0.4708**	0.3393	0.3363	0.3377	0.3491	0.3305	**0.3446**
No.15	0.4928	0.4605	0.4889	0.4937	0.4732	0.4836	0.3508	0.3477	**0.3498**	0.3485	0.3426	0.3459
No.16	0.4802	0.4708	0.4794	0.4896	0.4653	0.4777	0.3409	0.3372	0.3394	0.3430	0.3403	**0.3412**
No.17	0.4689	0.4324	0.4599	0.4758	0.4536	**0.4659**	0.3272	0.3240	0.3260	0.3262	0.3150	0.3224
No.18	0.4780	0.4418	0.4704	0.4771	0.4573	0.4683	0.3377	0.3348	0.3364	0.3472	0.3388	**0.3430**
No.19	0.4760	0.4448	0.4697	0.4814	0.4508	**0.4633**	0.3384	0.3358	0.3375	0.3388	0.3308	0.3343
No.20	0.4725	0.4658	0.4684	0.4773	0.4460	0.4669	0.3338	0.3304	0.3319	0.3349	0.3325	**0.3339**
No.21	0.4907	0.4513	0.4795	0.4722	0.4495	0.4631	0.3451	0.3421	0.3439	0.3539	0.3462	**0.3501**
No.22	0.4636	0.4407	0.4576	0.4605	0.4388	0.4522	0.3313	0.3282	0.3299	0.3430	0.3226	**0.3360**
No.23	0.4603	0.4299	0.4324	0.4663	0.4408	**0.4544**	0.3324	0.3267	0.3308	0.3332	0.3233	0.3291
No.24	0.5003	0.4575	0.4823	0.5104	0.4894	**0.5023**	0.3436	0.3405	0.3425	0.3524	0.3444	**0.3492**
No.25	0.4750	0.4734	0.4744	0.4844	0.4679	0.4760	0.3222	0.3184	0.3199	0.3224	0.3114	0.3178
No.26	0.4854	0.4527	0.4761	0.4841	0.4571	0.4698	0.3383	0.3357	0.3372	0.3472	0.3297	0.3390
No.27	0.4698	0.4495	0.4671	0.4642	0.4363	0.4516	0.3349	0.3316	0.3334	0.3350	0.3259	0.3308
No.28	0.4987	0.4963	**0.4976**	0.4916	0.4640	0.4785	0.3367	0.3343	0.3357	0.3375	0.3348	**0.3351**
No.29	0.4790	0.4705	**0.4768**	0.4714	0.4406	0.4591	0.3380	0.3349	0.3368	0.3410	0.3291	**0.3385**
No.30	0.4577	0.4322	0.4405	0.4503	0.4245	0.4378	0.3289	0.3236	0.3267	0.3325	0.3201	0.3248

V. CONCLUSIONS AND DISCUSSIONS

In this paper, we formulated two types of OTRAPs as multi-objective optimization problems: one with two objectives, and the other with three. Then, we solved the two problems using MOEAs. Specifically, a novel MOEA called HaD-MOEA was proposed because the state-of-the-art MOEA, namely NSGA-II, does not perform satisfactorily on the second type of OTRAP. Empirical studies were first carried out to evaluate the utility of MOEAs in comparison with existing single-objective approaches. MOEAs not only managed to achieve almost the same solution as that which can be attained by single-objective approaches, but also found simultaneously a set of alternative solutions. These solutions showed different trade-offs between the reliability of a system and the testing cost, and hence can facilitate informed planning of a testing phase. Later, a comprehensive experimental study was conducted to compare HaD-MOEA with NSGA-II on three parallel-series modular software systems, and a star-structure modular software system, each with a large variety of randomly generated parameter settings. The purpose was to evaluate the algorithms on systems with different sizes, structures, and parameters. The results showed that both algorithms performed well on the bi-objective problems, while HaD-MOEA performed significantly better than NSGA-II on the tri-objective problems, for which the total testing resource expenditure is not determined in advance. The superiority of HaD-MOEA consistently holds for all four tested systems.

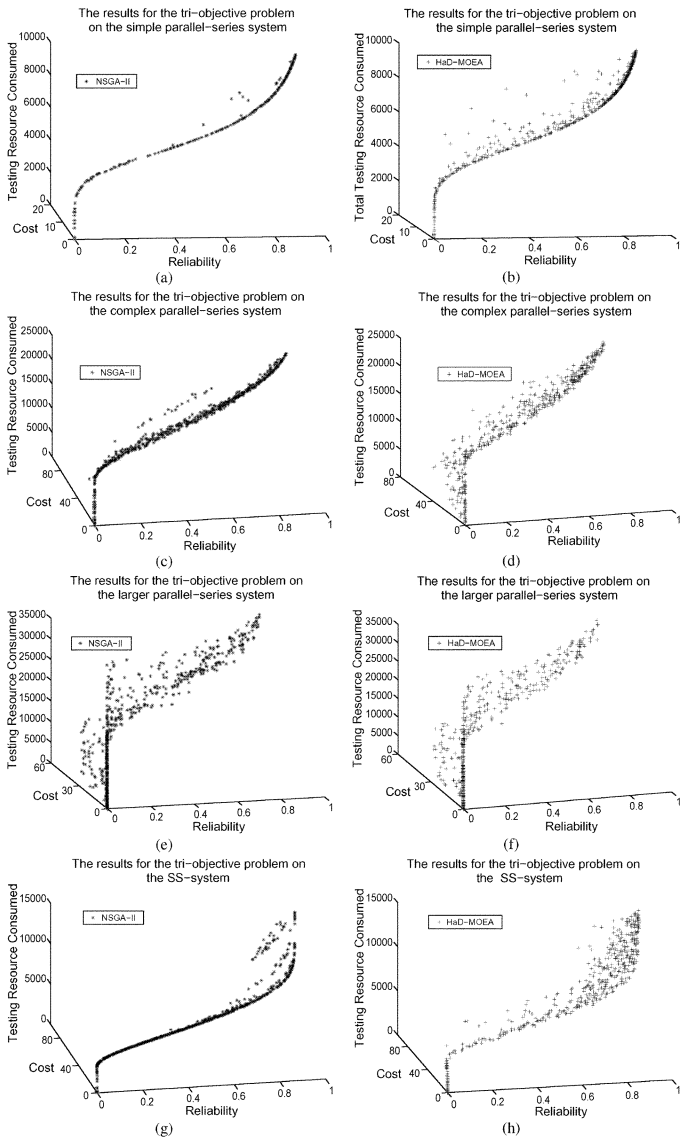

Fig. 11. Pareto sets obtained by NSGA-II, and HaD-MOEA for the tri-objective problems on the four employed software systems (one case study).

In addition to the above advantages, MOEAs may also be applied to real-world problems with little effort. Given a software testing task, a software tester only needs to formulate the reliability and cost as the functions of testing resource assigned to each modules of the system. Then, control parameters of the MOEAs can be set to those values used in this work, because they generally performed well in our empirical studies. After that, the algorithm can obtain a set of solutions (i.e., plans of testing resource assignment) without human involvement in the search process. The very modest computational cost of the MOEAs (in comparison with the total testing resource) also supports its potential utility in real-world scenarios.

Two directions may be further pursued in the future. First, the efficacy of MOEAs was only evaluated on parallel-series, and star-structure modular software systems; and the objective

TABLE XI
OVERALL COMPARISON BETWEEN HAD-MOEA AND NSGA-II FOR THE TWO MULTI-OBJECTIVE PROBLEMS ON FOUR SOFTWARE SYSTEMS (TWO-SIDED WILCOXON RANK-SUM TEST WITH A SIGNIFICANCE LEVEL 0.05 WAS USED). S, C, L AND SS STAND FOR SIMPLE, COMPLEX, LARGER AND SS SOFTWARE SYSTEMS, RESPECTIVELY. RESULTS ARE PRESENTED IN FORM OF WIN-DRAW-LOSE, INDICATING THE NUMBERS OF TEST CASES ON WHICH THE CORRESPONDING HAD-MOEA IS SUPERIOR, COMPARABLE (I.E., s-INSIGNIFICANT), AND INFERIOR TO THE NSGA-II

HaD-MOEA vs NSGA-II							
bi-objective problem				tri-objective problem			
S	C	L	SS	S	C	L	SS
10-9-11	7-20-3	6-20-4	6-18-6	22-6-2	14-13-3	22-8-0	15-13-2

TABLE XII
AVERAGE RUNTIME (IN CPU SECONDS) OF MOEAS ON THE FOUR CASE STUDIES

	bi-objective problem		tri-objective problem	
	NSGA-II	HaD-MOEA	NSGA-II	HaD-MOEA
simple system	20.7938	20.9200	113.1183	113.5321
complex system	22.0629	22.1821	124.0621	124.0681
larger system	22.9903	23.1542	136.0212	136.3032
SS-system	24.0784	24.3928	121.8521	121.9680

functions were formulated based on the assumption of s-independence between modules and components. However, in the reliability literature, some software systems are modeled under the assumption that the modules and components are dependent upon each other [27]. In this case, the objective functions (e.g., the reliability function) will be substantially different from those investigated in this work. It would be interesting to verify whether MOEAs can still work for such scenarios. Second, although HaD-MOEA was specifically designed to address the drawback of NSGA-II on the tri-objective problems, the improvement essentially arose from the consideration of multi-objective optimization. No domain-specific knowledge about OTRAPs was taken into account. However, it is commonly thought that incorporating domain-knowledge into an algorithm will further boost its performance on the specific problem. Hence, we will investigate designing novel search operators that are specifically suitable for OTRAPs.

REFERENCES

[1] H. Ohtera and S. Yamada, "Optimal allocation and control problems for software-testing resource," *IEEE Transactions on Reliablity*, vol. 39, no. 2, pp. 171–176, 1990.

[2] R. H. Huo, S. Y. Kuo, and Y. P. Chang, "Efficient allocation of testing resources for software module testing based on the hyper-geometric distribution software reliability growth model," *IEEE Transcations on Reliablity*, vol. 45, no. 4, pp. 541–549, 1996.

[3] O. Berman and N. Ashrafi, "Optimization models for reliability of modular software systems," *IEEE Transcations on Software Engineering*, vol. 19, no. 11, pp. 1119–1123, 1993.

[4] S. Yamada, T. Lehimori, and M. Nishiwaki, "Optimal allocation policies for testing resource based on a software reliability growth model," *Mathematical and Computer Modelling*, vol. 22, pp. 295–301, 1995.

[5] Y. W. Leung, "Dynamic resource allocation for software module testing," *The Journal of the Systems and Software*, vol. 37, no. 2, pp. 129–139, 1997.

[6] D. W. Coit, "Economic allocation of test times for subsystem-level reliability growth testing," *IIE Transcations on Business*, vol. 30, no. 12, pp. 1143–1151, 1998.

[7] D. W. Coit and A. Smith, "Reliability optimization of series-parallel systems using a genetic algorithm," *IEEE Transactions on Reliability*, vol. 45, no. 2, pp. 254–260, 1996.

[8] B. Yang and M. Xie, "Testing-resource allocation for redundant software systems," in *Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing (PRDC'99)*, Hong Kong, China, 1999, pp. 73–83.

[9] B. Yang and M. Xie, "A study of operational and testing reliability in software reliability analysis," *Reliability Engineering and System Safety*, vol. 70, pp. 323–329, 2000.

[10] B. Yang and M. Xie, "Optimal testing-time allocation for modular systems," *International Journal of Quality and Reliability Management*, vol. 18, no. 8, pp. 854–863, 2001.

[11] C. Y. Huang and M. R. Lyu, "Optimal testing resource allocation, and sensitivity analysis in software development," *IEEE Transactions on Reliability*, vol. 54, no. 4, pp. 592–603, 2005.

[12] Y. S. Dai, M. Xie, K. L. Poh, and B. Yang, "Optimal testing-resource allocation with genetic algorithm for modular software systems," *The Journal of Systems and Software*, vol. 66, pp. 47–55, 2003.

[13] M. R. Lyu, S. Rangarajan, and A. P. A. V. Moorsel, "Optimal allocation of testing resources for software reliability growth modeling in component-based software development," *IEEE Transactions on Reliability*, vol. 51, no. 2, pp. 183–192, 2002.

[14] D. W. Coit, T. Jin, and N. Wattanapongsakorn, "System optimization with component reliability estimation uncertainty: A multicriteria approach," *IEEE Transactions on Reliability*, vol. 53, no. 3, pp. 369–380, 2004.

[15] Y. S. Dai and X. L. Wang, "Optimal resource allocation on grid systems for maximizing service reliability using a genetic algorithms," *Reliability Engineering and System Safety*, vol. 91, no. 9, pp. 1071–1082, 2006.

[16] C. Y. Huang and J. H. Lo, "Optimal resource allocation for cost and reliability of modular software systems in the testing phase," *The Journals of Systems and Software*, vol. 79, pp. 653–664, 2006.

[17] P. Y. Yin and J. Y. Wang, "Ant colony optimization for the nonlinear resource allocation problem," *Applied Mathematics and Computation*, vol. 174, pp. 1438–1453, 2006.

[18] P. Y. Yin and J. Y. Wang, "A particle swarm optimization approach to the nonlinear resource allocation problem," *Applied Mathematics and Computation*, vol. 183, pp. 232–242, 2006.

[19] Y. S. Dai and G. Levitin, "Optimal resource allocation for maximizing performance and reliability in tree-structured grid services," *IEEE Transactions on Reliability*, vol. 56, no. 3, pp. 444–453, 2007.

[20] Y. S. Dai, G. Levitin, and X. Wang, "Optimal task partition and distribution in grid service system with common cause failures," *Future Generation Computer Systems*, vol. 23, no. 2, pp. 209–218, 2007.

[21] G. Levitin and Y. S. Dai, "Optimal service task partition and distribution in grid system with star topology," *Reliability Engineering and System Safety*, vol. 93, no. 1, pp. 153–160, 2008.

[22] S. J. Sadjadi and R. Soltani, "An efficient heuristic versus a robust hybrid meta-heuristic for general framework of serial-parallel redundancy problem," *Reliability Engineering and System Safety*, vol. 94, no. 11, pp. 1703–1710, 2009.

[23] Q. Long, M. Xie, S. Ng, and G. Levitin, "Reliability analysis and optimization of weighted voting systems with continuous states input," *European Journal of Operational Research*, vol. 191, no. 1, pp. 238–250, 2008.

[24] R. Tavakkoli-Moghaddam, J. Safari, and F. Sassani, "Reliability optimization of series-parallel systems with a choice of redundancy strategies using a genetic algorithm," *Reliability Engineering and System Safety*, vol. 93, no. 4, pp. 550–556, 2008.

[25] Z. Wang, K. Tang, and X. Yao, "A multi-objective approach to testing resource allocation in modular software systems," in *Proceedings of the 2008 IEEE Congress on Evolutionary Computation (CEC2008)*, Hongkong, China, 2008, pp. 1148–1153.

[26] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[27] W. Kuo and R. Wan, "Recent advances in optimal reliability allocation," *IEEE Transcations on Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol. 37, pp. 143–156, 2007.

[28] C. A. C. Coello, "Evolutionary multi-objective optimisation: An historical view of the field," *IEEE Computational Intelligence Magazine*, no. 1, pp. 28–36, 2006.
[29] A. Konak, D. W. Coit, and A. E. Smit, "Multi-objective optimization using genetic algorithms: A tutorial," *Reliability Engineering and System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.
[30] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms*, Pittsburgh, PA, USA, July 24–26, 1985, pp. 93–100.
[31] J. D. Knowles and D. W. Corne, "Approximating the nondominated front using the pareto archived evolution strategy," *Evolutionary Computation*, vol. 8, no. 2, pp. 149–172, 2000.
[32] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1994.
[33] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A niched pareto genetic algorithm for multiobjective optimization," in *Proceedings of the First IEEE Conference on Evolutionary Computation*, Orlando, FL, USA, June 27–29, 1994, vol. 1, pp. 82–87.
[34] C. M. Fonseca and P. J. Fleming, "Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, California, July 17–22, 1993, pp. 416–423.
[35] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 4, pp. 257–271, 1999.
[36] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization," in *Evolutionary Methods for Design, Optimisation and Control With Application to Industrial Problems (EUROGEN 2001)*, K. C. Giannakoglou, Ed. *et al.*, Athens, Greece, September 19-21, 2001, pp. 95–100.
[37] E. Zitzler and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, 2000.
[38] S. Huband, P. Hingston, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 477–506, 2006.
[39] C. M. Lin and M. Gen, "Multiobjective resource allocation problem by multistage decision-based hybrid genetic algorithm," *Applied Mathematics and Computation*, vol. 187, no. 2, pp. 574–583, 2007.
[40] R. C. Purshouse and P. J. Fleming, "On the evolutionary optimization of many conflicting objectives," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 770–784, 2007.
[41] P. A. Tom and C. S. R. Murthy, "Algorithms for reliability-oriented module allocation in distributed computing systems," *The Journal of System and Software*, vol. 40, no. 2, pp. 125–138, 1998.
[42] E. Zitler, L. Thiele, M. Laumanns, C. M. Foneseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.
[43] M. Fleischer, The Measure of Pareto Optima: Applications to Multi-Objective Metaheuristics Inst Systems Research, Univ. Maryland, College Park, MD, Tech. Rep. ISR TR 2002–32, 2002.

Zai Wang (S'08) received the B.S. degree in computer science from the School of Computer Science and Technology, University of Science and Technology of China (USTC), Hefei, China, in 2006, and is currently working toward the Ph.D. degree at the Nature Inspired Computation and Applications Laboratory (NICAL), School of Computer Science and Technology, USTC. He is the recipient of the 2010 Graduate Student Research Grants of IEEE Computational Intelligence Society.

Ke Tang (S'05–M'06) received the B.Eng. degree from the Huazhong University of Science and Technology, Wuhan, China, in 2002 and the Ph.D. degree from the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, in 2007.

Since 2007, he has been an Associate Professor with the Nature Inspired Computation and Applications Laboratory, School of Computer Science and Technology, University of Science and Technology of China, Hefei, China. He is also an Honorary Senior Research Fellow in the School of Computer Science, University of Birmingham, UK. He has authored/co-authored more than 40 refereed publications. His major research interests include machine learning, pattern analysis, evolutionary computation, data mining, metaheuristic algorithms, and real-world applications.

Dr. Tang is an Associate Editor of IEEE Computational Intelligence Magazine, editorial board member of three international journals, and the Chair of IEEE Task Force on Large Scale Global Optimization.

Xin Yao (M'91–SM'96–F'03) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, China, in 1982, the M.Sc. degree from the North China Institute of Computing Technology, Beijing, China, in 1985, and the Ph.D. degree from USTC in 1990.

From 1985 to 1990, he was an Associate Lecturer and Lecturer with USTC, while working toward the Ph.D. degree in simulated annealing and evolutionary algorithms. In 1990, he was a Postdoctoral Fellow with the Computer Sciences Laboratory, Australian National University, Canberra, Australia, where he continued his work on simulated annealing and evolutionary algorithms. In 1991, he was with the Knowledge-Based Systems Group, Commonwealth Scientific and Industrial Research Organization, Division of Building, Construction and Engineering, Melbourne, Australia, where he worked primarily on an industrial project on automatic inspection of sewage pipes. In 1992, he returned to Canberra to take up a lectureship in the School of Computer Science, University College, University of New South Wales, Australian Defense Force Academy, Sydney, Australia, where he was later promoted to a Senior Lecturer and Associate Professor. Since April 1999, he has been a Professor (Chair) of computer science in the University of Birmingham, Birmingham, U.K. He is currently the Director of the Center of Excellence for Research in Computational Intelligence and Applications, School of Computer Science, University of Birmingham, Birmingham, U.K. and also a Changjiang (Visiting) Chair Professor (Cheung Kong Scholar) with the Nature Inspired Computation and Applications Laboratory, School of Computer Science and Technology, USTC. He has given more than 50 invited keynote and plenary speeches at conferences and workshops worldwide. He has more than 300 referenced publications. His major research interests include evolutionary artificial neural networks, automatic modularization of machine learning systems, evolutionary optimization, constraint-handling techniques, computational time complexity of evolutionary algorithms, coevolution, iterated prisoner's dilemma, data mining, and real-world applications.

Dr. Yao was the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION from 2003 to 2008, an Associate Editor or editorial board member of 12 other journals, and the Editor of the World Scientific Book Series on Advances in Natural Computation. He was the recipient of the President's Award for Outstanding Thesis by the Chinese Academy of Sciences for his Ph.D. work on simulated annealing and evolutionary algorithms in 1989. He was the recipient of the 2001 IEEE Donald G. Fink Prize Paper Award for his work on evolutionary artificial neural networks.