The Journal of Systems and Software 000 (2014) 1–22

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Cost-effective test suite minimization in product lines using search techniques

Shuai Wang[a,b,*], Shaukat Ali[a], Arnaud Gotlieb[a]

[a] Certus Software V&V Center, Simula Research Laboratory, P.O. 134, 1325 Lysaker, Oslo, Norway
[b] Department of Informatics, University of Oslo, P.O. 1080, 0316 Blindern, Oslo, Norway

ARTICLE INFO

Article history:
Received 1 December 2013
Revised 9 June 2014
Accepted 16 August 2014
Available online xxx

Keywords:
Product line
Search algorithm
Test suite minimization

ABSTRACT

Cost-effective testing of a product in a product line requires obtaining a set of relevant test cases from the entire test suite via test selection and minimization techniques. In this paper, we particularly focus on test minimization for product lines, which identifies and eliminates redundant test cases from test suites in order to reduce the total number of test cases to execute, thereby improving the efficiency of testing. However, such minimization may result in the minimized test suite with low test coverage, low fault revealing capability, low priority test cases, and require more time than the allowed testing budget (e.g., time) as compared to the original test suite. To deal with the above issues, we formulated the minimization problem as a search problem and defined a fitness function considering various optimization objectives based on the above issues. To assess the performance of our fitness function, we conducted an extensive empirical evaluation by investigating the fitness function with three weight-based Genetic Algorithms (GAs) and seven multi-objective search algorithms using an industrial case study and 500 artificial problems inspired from the industrial case study. The results show that Random-Weighted Genetic Algorithm (RWGA) significantly outperforms the other algorithms since RWGA can balance all the objectives together by dynamically updating weights during each generation. Based on the results of our empirical evaluation, we also implemented a tool called TEst Minimization using Search Algorithms (TEMSA) to support test minimization using various search algorithms in the context of product lines.

1. Introduction

Product Line Engineering (PLE) has proven to be a cost-effective way to develop similar products by exploiting and managing commonalities and variabilities among a large number of products (Benavides et al., 2010; McGregor, 2001). PLE has shown promising benefits in both academia and industry such as reducing development time and costs, speeding up product time-to-market and improving the quality of products of a product line family (Benavides et al., 2010). Cost-effective testing of a new product in a product line can be achieved by systematic application of model-based PLE approaches (McGregor, 2001), for example, based on feature models (Benavides et al., 2010). Feature models have become the de-facto standard for capturing commonalities and variabilities in product lines and can be used for systematic selection of test cases for a new product in a product line (Benavides et al., 2010).

An overall view of our product line testing project is presented in Fig. 1. The project has two main activities: (1) test selection for a product in a product line and (2) test minimization for the selected test cases following the first activity. Test selection was focused in our previous works (Wang et al., 2012, 2013a), where we proposed a methodology for automated selection of test cases for a new product using Feature Model (FM) and Component Family Model (CFM) (GmbH, 2006). FM was used to capture commonalities and variabilities of features of a product line, where CFM was used to model the structure of test cases in a test case repository as shown in Fig. 1. Two types of traceability links were established: (1) traceability links (traces) from CFM to actual test cases in the repository; (2) links between FM and CFM using pre-defined restrictions. To test a new product (e.g., $Product_y$ as shown in Fig. 1), a test engineer only requires performing simple selection of relevant features in the FM and the corresponding test cases can be automatically obtained from the test case repository.

Based on our experience of working with Cisco, we discovered that even after test selection, the number of test cases is still large (more that 2000 requiring 3–4 days to execute per product) and many of the test cases are redundant (e.g., several test cases cover the same functionality). On top of that, new test cases

* Corresponding author at: Certus Software V&V Center, Simula Research Laboratory, P.O. 134, 1325 Lysaker, Oslo, Norway. Tel.: +47 45239921.
E-mail addresses: shuai@simula.no (S. Wang), shaukat@simula.no (S. Ali), arnaud@simula.no (A. Gotlieb).

http://dx.doi.org/10.1016/j.jss.2014.08.024
0164-1212/

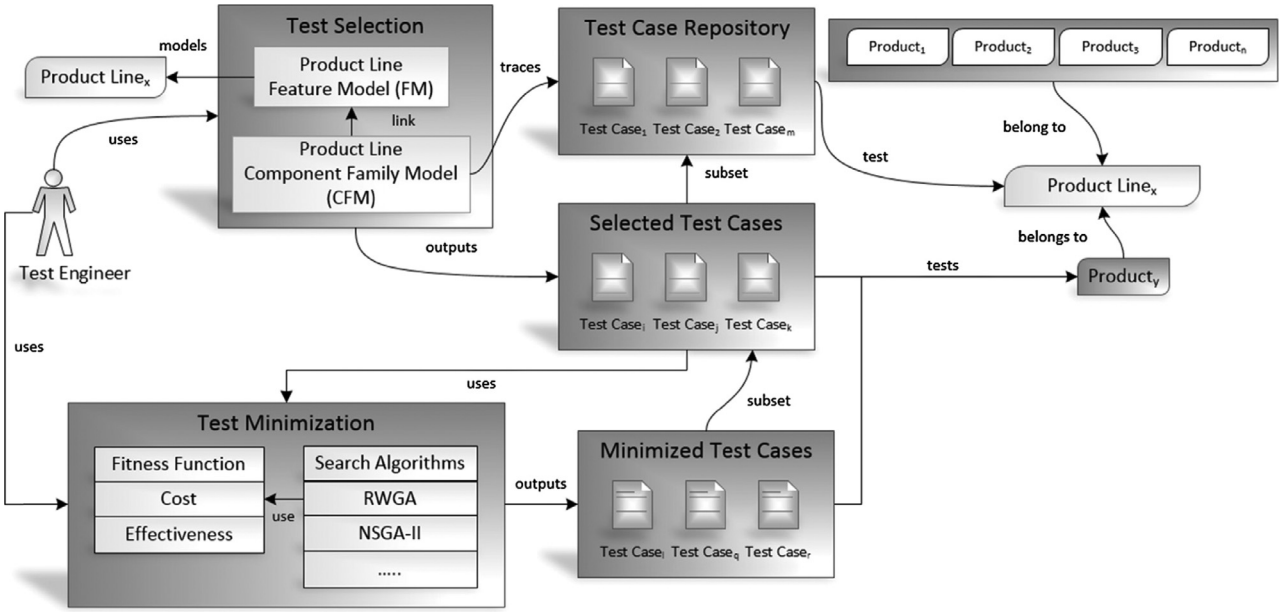


Fig. 1. An overview of cost-effective testing of products in a product line.

for product lines are continuously added further increasing the number of test cases. Executing all these test cases is practically impossible within the allocated testing time and thus warrants efficient and sophisticated test minimization techniques (Engström and Runeson, 2011; Harman, 2011; Lopez-Herrejon et al., 2013; Runeson and Engström, 2012; Wang et al., 2013c). However, such minimization may reduce cost (e.g., execution time), but on the other hand may also reduce effectiveness (e.g., fault detection capability). This means an efficient technique must achieve a desirable balance among cost and effectiveness measures, which is a multi-objective optimization problem. A number of such objectives have been studied for multi-objective test optimization in regression testing in the existing literature such as execution time, test coverage and fault detection capability (Harman, 2011; Yoo and Harman, 2012).

Multi-objective search algorithms are well adapted for the type of problem we are solving in this paper (Harman et al., 2009; Konak et al., 2006). Moreover, Weight-based Genetic Algorithms (GAs) are also known to solve multi-objective optimization problems by assigning a set of weights to each objective (Konak et al., 2006). In our previous work (Wang et al., 2013c), we defined a fitness function based on the following cost/effectiveness measures based on our industrial collaboration and the existing literature (Engström and Runeson, 2011; Harman, 2011; Lopez-Herrejon et al., 2013): (1) *Cost*: minimizing the number of test cases thereby indirectly reducing execution cost referred as Test Minimization Percentage (*TMP*); (2) *Effectiveness*: High Feature Pairwise Coverage (*FPC*) and Fault Detection Capability (*FDC*). In Wang et al. (2013c), we compared three weight-based Genetic Algorithms (GAs) and showed that Random-Weighted GA (RWGA) had significantly better performance than the other algorithms.

Further investigation at Cisco revealed the following additional cost and effectiveness measures: (1) *Cost*: overall execution cost of the minimized test cases; (2) *Effectiveness*: priority of test cases. Notice these two additional measures have also been investigated in the existing literatures (Engström and Runeson, 2011; Harman, 2011; Lopez-Herrejon et al., 2013). Moreover, we also decided to compare weight-based algorithms with other types of algorithms such as Fast Non-dominated Sorting Genetic Algorithm (NSGA-II) (Konak et al., 2006; Yoo and Harman, 2007). Based on these observations, this paper extends our existing work (Wang et al., 2013c) by including: (1) four effectiveness measures, i.e., Test Minimization Percentage (*TMP*), Pairwise Coverage (*FPC*), Fault Detection Capability (*FDC*) and Average Execution Frequency (*AEF*) and one cost measure, i.e., Overall Execution Time (*OET*); (2) multi-objective fitness function considering all the proposed cost/effectiveness measures; and (3) empirical evaluation for the proposed fitness function in conjunction with three weight-based multi-objective search algorithms and seven other multi-objective search algorithms on an industrial case study and assessing the scalability using 500 artificial problems of varying complexity; (4) a tool called TEst Minimization with Search Algorithms (TEMSA) based on Durillo and Nebro (2011).

The results of our empirical evaluation showed that: (1) based on the industrial case study, all the selected search algorithms except Improved Strength Pareto Evolutionary Algorithm (SPEA2) are cost-effective for solving the test minimization problem and RWGA achieves the best performance; (2) for the 500 artificial problems, the results are consistent with the industrial case study, i.e., RWGA outperforms all the other search algorithms. Furthermore, RWGA managed to solve the problems of varying complexity.

The rest of the paper is organized as follows. Section 2 provides a brief introduction of the selected search algorithms. Section 3 presents a formal representation of our problem, definitions and functions for the cost/effectiveness measures and fitness function used by all the algorithms. Section 4 describes our industrial case study and designed artificial problems followed by empirical evaluation of the applied algorithms in Section 5. Section 6 discusses the results and analysis for the experiments and Section 7 presents an overall discussion based on the obtained results. In Section 8, the tool support is discussed. Section 9 addresses the threats to validity of our empirical study and related work is discussed in Section 10. Section 11 concludes the paper and presents future work.

2. Description of the selected algorithms

In this section, we provide a brief description of the ten algorithms that were selected for our experiments. These algorithms are classified into four working mechanisms to guide the search

Table 1
Classification of the selected search algorithms.

Different mechanisms			Algorithms	Minimization before?
Evolutionary Algorithms (EAs)	GAs	Weight-Based GA	WBGA	Yes
			WBGA-MO	Yes
			RWGA	Yes
		Sorting-Based GA	NSGA-II	Yes
		Cellular-Based GA	MOCell	No
	Strength Pareto EA		SPEA2	Yes
	Evolution Strategies		PAES	No
Swarm Algorithm	Particle Swarm Optimization		SMPSO	No
Hybrid Algorithm	Cellular genetic algorithm + differential evolution		CellDE	No
Stochastic Algorithm	Random Search		RS	Yes

(Brownlee, 2012), which are: Evolutionary Algorithms (EAs), Swarm Algorithms, Hybrid Algorithms and Stochastic Algorithms (Table 1). The column *Minimization Before* shows whether the listed algorithms have been adapted for solving the test minimization problem in the existing literature. Notice that one of our aims is to evaluate a representative set of the existing search algorithms rather than limiting us to the commonly used algorithms (e.g., NSGA-II and SPEA2). To achieve this goal, we chose ten search algorithms systematically based on the categories classified in (Brownlee, 2012). Before getting into more details for each algorithm (Sections 2.2–2.5), we first provide a brief overview for search algorithms (Section 2.1).

2.1. Overview of search algorithms

Generally speaking, search algorithms aim at mimicking natural phenomenon, for example, natural evolution process or behaviors of animals (e.g., bird flocking) to search optimal solutions for various optimization problems (Brownlee, 2012). To apply search algorithms, fitness function (objective function) should be defined mathematically to guide the search with aims at assessing the capability of the solutions.

Moreover, for genetic algorithms (GAs), genes, individuals (also called chromosomes) should be carefully defined beforehand. More specifically, individuals refer to potential solutions with respect to the optimization problem and each individual (solution) is composed of a set of genes, which present units for the solution, e.g., test cases for the test suite in our case. The set of individuals handled by GAs is represented as population. Similarly, for swarm algorithm (Section 2.3), the potential solutions are named as particles and the set of the solutions handled by algorithms refers to swarm.

After properly defining fitness function (objective function), the search algorithms will be repeatedly run by applying various operators (e.g., selection) to produce the best individual (solution) until the terminating condition is satisfied (e.g., number of fitness evaluation). Three operators are usually adapted by various search algorithms, namely selection, crossover (also called recombination) and mutation (Brownlee, 2012). More specifically, (1) *Selection* operator corresponds to the operation of selecting the solutions (e.g., individuals for evolutionary algorithms) with the best values based on the defined fitness function. Notice that the *Selection* operator can be applied to all the search algorithms; (2) *Crossover* operator is mainly applied in various GAs and refers to the operation of selecting two individuals (usually called parent individuals) and exchanging their genes partially. The genes that will be switched between the individuals are usually determined at random. New individuals obtained by crossover operator are usually called offspring; and (3) *Mutation* operator is also mostly adapted in GAs and refers to updating the individuals by changing the properties of genes randomly based on the predefined mutation rate (i.e., probability of changing the properties of genes).

2.2. Evolutionary Algorithms (EAs)

EAs are inspired by biological evolution, which was first proposed by Darwin to describe how natural selection and mutation occurs on various genes of species (Brownlee, 2012). Since EAs are well known in search-based software engineering (Harman et al., 2009), we choose seven of them, which are further classified into GAs, Strength Pareto EA and Evolution Strategies. We describe each of them in detail as below.

The working theory of these three weight-based GAs is to assign a particular weight to each objective function for converting the multi-objective problem to a single objective problem with a scalar objective function (Konak et al., 2006). The difference of these algorithms can be summarized as the different mechanisms of assigning weights for each objective. Weight-Based GA (WBGA) assigns fixed weights for each objective defined for each objective beforehand, which can be provided by users based on the domain knowledge and expertise. The pre-determined weight for each objective is used during all generations of the algorithm. WBGA for Multi-objective Optimization (WBGA-MO) uses a weight pool including a set of predefined weights and each solution can choose weights for each objective randomly at each generation. The assigned weight for each objective changes during each generation, but is still limited to the ones in the weight pool. Notice that such weight pool can also be pre-provided by users. RWGA randomly assigns normalized weights to multiple objective functions for each solution when selecting fittest individuals at each generation, i.e., assigning weights to objective functions is dynamic and there is no need to set fixed weights for different objectives beforehand.

NSGA-II is based on the Pareto dominance theory, which outputs a set of non-dominated solutions for multiple objectives (Deb et al., 2002). The population is sorted into several non-dominated fronts using a ranking algorithm first. Then, individual solutions are selected from these non-dominated fronts by calculating the *crowd distance*, which is used to measure the distance between the individual solutions and the rest of the solutions in the population (Konak et al., 2006). If two individual solutions are in the same non-dominated front, the solution with a higher value of crowd distance will be selected.

Multi-objective Cellular (MOCell) is based on the cellular model of GAs (cGAs) with an assumption that an individual only interact with its neighbors during the search process (*neighborhood*) (Nebro et al., 2007). Moreover, MOCell stores a set of obtained non-dominated individual solutions in an external archive. After each generation, MOCell replaces a fixed number of solutions chosen randomly from the population by selecting the same number of solutions from the archive until the termination conditions are met. Such replacement only takes place when the newly generated solutions from the population are worse than the solutions in the archive.

Improved Strength Pareto Evolutionary Algorithm (SPEA2) is also based on Pareto dominance theory (Zitzler et al., 2001). The fitness

Please cite this article as: S. Wang et al., Cost-effective test suite minimization in product lines using search techniques, The Journal of Systems and Software (2014), http://dx.doi.org/10.1016/j.jss.2014.08.024

value for each solution is calculated by summing up a strength raw fitness based on the defined objective functions and density estimation, where density estimation measures the distance between a solution and its nearest neighbors for maximizing the diversity of the obtained solutions. SPEA2 first stores a fixed number of best solutions into an archive by applying selection, crossover and mutation operators. Then, a new population is created by combining the solutions from the archive and the non-dominated solutions of the original solution. If the combined non-dominated solutions are more than the maximum size of population, the solution with minimum distance to any other solutions is selected by applying a truncation operator to calculate the distances to the its neighborhood.

Pareto Archived Evolution Strategy (PAES) is inspired by evolution theory (Knowles and Corne, 2000). By applying dynamic crossover, mutation operators, PAES aims at maximizing the suitability for the selected solutions. Moreover, PAES stores the obtained non-dominated solutions into an archive and newly generated solutions can be added into such archive if they are better than the existing solutions by calculating objective functions.

2.3. Swarm Algorithm

Particle Swarm Optimization (PSO) is a biological metaheuristic inspired by the social foraging behavior of animals such as bird flocking (Brownlee, 2012).

Speed-constrained Multi-objective Particle Swarm Optimization (SMPSO) algorithm is based on PSO theory (Knowles and Corne, 2000; Nebro et al., 2009). Similar as NSGA-II, SMPSO selects best solutions by calculating crowding distance and also stores the selected individual solutions in an archive. SMPSO takes advantage of mutation operators to accelerate the speed of convergence and adapts velocity constriction mechanism to avoid the explosion of swarms. According to Knowles and Corne (2000) and Nebro et al. (2009), the performance of SMPSO is considered as the best one among the existing PSO algorithms.

2.4. Hybrid Algorithm

Differential Evolution (DE) algorithm is considered as another kind of EA, which generates solutions by applying recombination, mutation, and selection operators (Brownlee, 2012). More specifically, DE calculates the weighted difference between two randomly selected solutions and integrate the obtained parts into a third solution for generating a new solution.

CellDE is a hybrid metaheuristic algorithm using MOCell as a search engine and replacing the typical selection, crossover and mutation operators for GAs with the recombination mechanism of DE (Durillo et al., 2008). It is considered that CellDE takes the advantage of cellular GA and DE with good diversity and convergence and the performance of CellDE improves significantly as compared with separate ones (Durillo et al., 2008).

2.5. Stochastic Algorithm

Random Search (RS) is a stochastic algorithm and is commonly used as a baseline for evaluation (Brownlee, 2012). RS is adapted to randomly generate solutions from the entire search space during each generation. Notice each newly generated solution is independent of the obtained solutions.

3. Problem representation and fitness function

To guide the search toward an optimal solution using multi-objective search algorithms, it is essential to define a fitness function, as is the case for any search algorithm. In Section 3.1, we present a formal representation of the search problem, followed by definitions and functions for the cost/effectiveness measures in Section 3.2. In Section 3.3, the overall fitness function is defined.

3.1. Problem representation

To precisely define our test minimization problem, the following definitions are presented first.

3.1.1. Basic concepts

Let $P = \{p_1, p_2, p_3, \ldots, p_{np}\}$ be a product line with a set of products, where np is the number of products in P.

$F = \{f_1, f_2, f_3, \ldots, f_{nf}\}$ is a feature model with a set of features to represent P (Benavides et al., 2010), where nf is the number of features, i.e., functionalities need to be tested for P.

$TS = \{t_1, t_2, t_3, \ldots, t_{nt}\}$ is a test suite including a large number of test cases (nt) for testing P.

$F_{p_i} = \{f_1, f_2, f_3, \ldots, f_{nf_{p_i}}\}$ is a subset of F to represent a specific product p_i which is to be tested (Benavides et al., 2010), where f_i can be any feature in F ($f_i \in F$) and nf_{p_i} is the number of features of F_{p_i} ($1 \leq nf_{p_i} \leq nf$).

$TS_{p_i} = \{t_1, t_2, t_3, \ldots, t_{nt_{p_i}}\}$ is a subset of the test suite TS for testing the p_i product, where t_i can be any test case in TS ($t_i \in TS$) and $1 \leq nt_{p_i} \leq nt$. Notice in our case, TS_{p_i} is obtained by our pervious test case selection methodology proposed in Wang et al. (2012, 2013a). Notice that the proposed approach in this paper can be applied separately (without the test case selection methodology) for multi-objective test minimization problems in other contexts.

$S_{p_i} = \{S_{p_i1}, S_{p_i2}, S_{p_i3}, \ldots, S_{ns_{p_i}}\}$ is a set of potential solutions based on TS_{p_i} for p_i, where ns_{p_i} is the total number of solutions for testing p_i, which can be measured as $2^{nt_{p_i}} - 1$. Notice that each potential solution includes a set of test cases from TS_{p_i}. As the number of test cases increases, the potential solutions increase exponentially. Suppose we have 1000 test cases to test the product line P and using the methodology proposed by Wang et al. (2012, 2013a), 100 test cases are obtained to test the product p_i. Then there will be $2^{100} - 1$ potential solutions for testing p_i, whose solution space is huge.

3.1.2. Cost/effectiveness measures

$Cost = \{Cost_1, Cost_2, \ldots, Cost_{\text{ncost}}\}$ is a set of cost measures for minimization, e.g., execution time, cost of a test minimization algorithm.

$Cost(s_i, CostMeasure)$ is a function, which returns the cost of a solution s_i from S_{p_i} based on a cost measure ($CostMeasure$) from $Cost$.

$Effect = \{effect_1, effect_2, \ldots, effect_{neffect}\}$ is a set of effectiveness measures for minimization, e.g., test minimization percentage, feature pairwise coverage.

$Effect(s_i, EffectivenessMeasure)$ is a function, which returns the effectiveness of a solution s_i from S_{p_i} based on an effectiveness measure ($EffectivenessMeasure$) from $Effect$.

Each solution s_i in S_{p_i} comprised nt_{s_i} test cases from TS_{p_i}: $\{t_{s_i1}, t_{s_i2}, t_{s_i3}, \ldots, t_{nt_{s_i}}\}$, where $1 \leq nt_{s_i} \leq nt_{p_i}$ and has a set of values for cost measures $Cost(s_i, CostMeasure)$ and effectiveness measures $Effect(s_i, EffectivenessMeasure)$, which are defined in the next section.

3.1.3. Problem representation

Our optimization problem can be represented as:

Problem. Search for a solution s_k (s_k comprised $\{t_{s_k1}, t_{s_k2}, t_{s_k3}, \ldots, t_{nt_{s_k}}\}$ from TS_{p_i}, where $1 \leq nt_{s_k} \leq nt_{p_i}$) from

S_{p_i} for testing the product p_i to achieve the following two objectives (i.e., maximum effectiveness and minimum cost):

$\forall s_l \in S_{p_i} \cap s_l \neq s_k$:

$$\sum_{j=1}^{neffect} Effect(s_k, effect_j) \geq \sum_{j=1}^{neffect} Effect(s_l, effect_j)$$

$$\sum_{j=1}^{ncost} Cost(s_k, cost_j) \leq \sum_{j=1}^{ncost} Cost(s_l, cost_j)$$

3.2. Definitions and functions for effectiveness/cost measures

In this section, based on the above-mentioned problem, we provide mathematical definitions and functions of relevant effectiveness and cost measures in our context.

3.2.1. Effectiveness measures

In our context, effectiveness measures *Effect* include four elements, i.e., test minimization percentage (*TMP*), feature pairwise coverage (*FPC*), fault detection capability (*FDC*) and average execution frequency (*AEF*). We define them in detail below:

3.2.1.1. Test minimization percentage (TMP). Definition 1. TMP is to measure the amount of reduction for the number of test cases.

Objective Function 1. TMP is calculated as follows:

$$TMP_{s_k} = \left(1 - \frac{nt_{s_k}}{nt_{p_i}}\right) \times 100\%$$

As discussed in Section 3.1, nt_{s_k} is the number of test cases for the specific solution s_k obtained by search algorithms for testing the product p_i, where $1 \leq nt_{s_k} \leq nt_{p_i}$. nt_{p_i} is the number of test cases in the test suite for testing product p_i. *TMP* value ranges from 0 to 1 and a higher value of *TMP* shows more test cases are reduced in the minimized test suite.

3.2.1.2. Feature pairwise coverage (FPC). Definition 2. FPC is to measure how much pairwise coverage can be achieved by a chosen solution (Kuo-Chung and Yu, 2002).

We chose this type of coverage based on our domain knowledge, discussion with test engineers, and history data about faults since a higher percentage of detected faults are mainly due to the interactions between features. Notice that other types of coverage (e.g., code coverage) can also be introduced into the fitness function based on other cases. *FPC* in our case is designed to compute the capability of covering feature pairs by a chosen solution.

Objective Function 2. The function for measuring *FPC* is shown as below.

$$FPC_{s_k} = \frac{Num_{FP_{s_k}}}{Num_{FP_{p_i}}}$$

$Num\_FP_{s_k}$ is the number of feature pairs covered in the test cases for the solution s_k measured as follows.

$$Num\_FP_{s_k} = \sum_{i=1}^{nt_{s_k}} Num\_FP_{tc_i}$$

nt_{s_k} is the number of test cases for the solution s_k, where $1 \leq nt_{s_k} \leq nt_{p_i}$. $Num\_FP_{tc_i}$ is the number of unduplicated feature pairs covered by the test case $i(tc_i)$. The feature pairs covered by tc_i can be computed as: $Num\_FP_{tc_i} = C_2^{size(F_{tc_i})} \cdot size(F_{tc_i})$ is the number of features tested by test case tc_i. For instance, test case tc_i is used to test three features. Then, the feature pairs covered by test case tc_i are $C_2^3 = 3 \times 2/2 = 3$. Notice that if some feature pairs are repeated ones compared with the feature pairs covered by the previous test cases, repeated pairs will be removed when computing $Num_{FP_{s_k}}$. Meanwhile, feature pairs only refer to the valid feature pairs that are not violated against the constraints defined among various features. In our case, there are no specific constraints that are defined on the features. But if such constraints exist in our contexts, the invalid feature pairs will be eliminated when calculating $Num_{FP_{s_k}}$ and $Num_{FP_{p_i}}$.

$Num\_FP_{p_i}$ is all number of feature pairs for testing the product p_i measured as: $Num\_FP_{p_i} = C_2^{size(F_{p_i})} = nt_{p_i} \times (nt_{p_i} - 1)/2$. F_{p_i} is the set of features representing the product p_i including nt_{p_i} features. For instance, if p_i is represented by ten features, all feature pairs covered by p_i are $C_2^{10} = 45$. Notice *FPC* is calculated for a chosen test solution ranging from 0 to 1 and a higher value shows better feature pairwise coverage.

3.2.1.3. Fault detection capability (FDC). Definition 3. FDC is to measure the fault detection capability of a selected test solution for a product.

In our context, fault detection refers to the rate of successful execution for a test case in a given time, e.g., a week in our case. More specifically, the execution of a test case can be defined as a *success* if it can detect faults in a given time (a week in our case) and as a *fail* if it does not detect any fault.

Objective Function 3. FDC is measured using the following function.

$$FDC_{s_k} = \frac{\sum_{i=1}^{nt_{s_k}} SucR_{tc_i}}{nt_{s_k}}$$

s_k refers to a solution for a product including a set of test cases and nt_{s_k} is the number of test cases for the solution s_k, where $1 \leq nt_{s_k} \leq nt_{p_i}$. $SucR_{tc_i}$ is the successful rate of execution for test case tc_i, which can be measured as below.

$$SucR_{tc_i} = \frac{NumSuc_{tc_i}}{NumSuc_{tc_i} + NumFail_{tc_i}}$$

$NumSuc_{tc_i}$ is the number of *success* for test case tc_i and $NumFail_{tc_i}$ is the number of *fail* for test case tc_i. For instance, a test case is usually executed 1000 times per week in Cisco. So if it executes successfully for 800 times, the *SucR* is 800/1000 = 0.8.

Note that *FDC* value also ranges from 0 to 1 and a higher value represents better fault detection capability.

3.2.1.4. Average execution frequency (AEF). Definition 4. AEF is to measure the average execution frequency of the solution s_k based on the execution frequency of each included test cases during a given time, e.g., a week.

In our context, we found that a test suite may have high priority if it is executed very frequently during the given time based on the domain knowledge. Therefore, the minimized test suite should preferably consist of test cases with high priority, which can be determined by the average execution frequency. This is also our main objective to seek a solution with higher average execution frequency from the solution space.

Objective Function 4. The function to measure *AEF* is presented as below.

$$AEF_{s_k} = \frac{\sum_{i=1}^{nt_{s_k}} EF_{tc_i}}{nt_{s_k}}$$

nt_{s_k} is the number of test cases for the solution s_k, where $1 \leq nt_{s_k} \leq nt_{p_i}$. EF_{tc_i} is the execution frequency for the test case tc_i in a given week. For instance, suppose the solution s_k consists of three

test cases: tc_1, tc_2 and tc_3. These three test cases are executed 5 times, 6 times and 7 times, respectively. Then *AEF* for the solution s_k will be 6 (i.e., (5 + 6 + 7)/3 = 6). Notice that a higher value of *AEF* represents higher average execution frequency thereby higher priority.

3.2.2. Cost measures

Cost measures *Cost* includes one element in our context, i.e., overall execution time (*OET*), which is defined as below.

3.2.2.1. Overall execution time (OET). Definition 5. OET is to measure the overall execution time for the solution s_k consisting of a set of test cases.

Objective Function 5. OET can be measured using the function as below.

$$OET_{s_k} = \sum_{i=1}^{nt_{s_k}} AET_{tc_i}$$

nt_{s_k} is the number of test cases for the solution s_k, where $1 \leq nt_{s_k} \leq nt_{p_i}$. AET_{tc_i} is the average execution time for test case tc_i in the solution s_k, which can be calculated as below.

$$AET_{tc_i} = \frac{\sum_{i=1}^{EF_{tc_i}} ET_{tc_i}}{EF_{tc_i}}$$

EF_{tc_i} is the execution frequency of the test case tc_i in a given week and ET_{tc_i} is the overall time for each execution. For instance, the frequency of the test case tc_i is three in a given week and the time for each execution is 15 min, 20 min and 25 min. Then the $AET_{tc_i} = (15 + 20 + 25)/3 = 20$ min. Notice that a higher value of OET_{s_k} shows more time is required for the solution s_k to execute and our goal is to keep such time within the allowed time budget.

In summary, our test minimization problem is formulated as a five-objective optimization problem, which can be solved using search algorithms.

3.3. Fitness function

Based on the defined effectiveness measures and cost measures in the previous section, our fitness function is presented as below. Notice that a lower value of fitness function represents better performance for a solution.

$$Fitness = f(TMP, FPC, FDC, OET, AEF)$$

Notice that the obtained values by different objective functions do not stay at the same comparable level. For instance, *FDC* ranges from 0 to 1 and the value of *OET* can be 1.5 (h). So we first normalize the obtained values of all the five functions using the following normalization function (Greer and Ruhe, 2004; Zhang et al., 2008).

$$nor(x) = \frac{x}{x+1}$$

where x can be any value obtained for each objective (i.e., *TMP*, *FPC*, *FDC*, *OET* and *AEF*), such as 0.56 for *FDC* and 0.5 h for *OET*. Therefore, the specific fitness function for the weight-based GAs (i.e., WBGA, WBGA-MO and RWGA) can be formulated as below, which ranges from 0 to 1, where lower the value is, better the fitness function is.

$$Fitness\_w = 1 - w_1 \times nor(TMP) - w_2 \times nor(FPC) - w_3 \times nor(FDC) - w_4 \times (1 - nor(OET)) - w_5 \times nor(AEF)$$

w_1, w_2, w_3, w_4 and w_5 are the weights assigned to *TMP*, *FPC*, *FDC*, *OET* and *AEF*, respectively satisfying the following constraint: $\sum_{i=1}^{5} w_i = 1$. Using this way, multi-objective optimization problem is converted to a single objective problem with a scalar objective function, which is a classical approach and is efficient to be solved using GAs (Konak et al., 2006). Notice that RWGA assigns weights to objectives dynamically during each generation and it is not required to set fixed weights to different objectives before.

The other search algorithms are based on different mechanisms and there is no need to assign particular weights to each objective function. All the cost/effectiveness measures including five objectives with functions have been clearly defined in Section 3.2. Notice that using NSGA-II, MOCell, SPEA2, PAES, SMPSO, CellDE, a set of individual solutions can be obtained after a specific number of fitness generations.

Table 2
Four products in *Saturn*.

Product	# features	# test cases
C20	17	138
C40	25	167
C60	32	192
C90	43	239

4. Industrial case study and artificial problems

First, we used an industrial case study of Videoconferencing System (VCSs) product line called *Saturn* provided by Cisco Systems, Norway to evaluate our fitness function (CiscoSystems, 2010). *Saturn* has several VCS products including C20 and C90 and has more than 2000 test cases. Notice that not all test cases are relevant for each product and new test cases are developed for *Saturn* every day. Based on our collaboration with Cisco, we observed that the current testing practice is performed in two ways: (1) executing all the test cases for a product; (2) randomly selecting a subset of test cases for a product. Therefore, random search could be considered as an equivalent case, though not a very accurate measure and require additional experiments involving real testers, which can be performed in the future. The following paper will evaluate the performance of the selected search algorithms as compared with RS.

We chose four products C20, C40, C60 and C90 from *Saturn*. There are 169 features in *Saturn* and each product includes a subset of these features. Meanwhile, each feature can be tested by at least one test case (usually more than one). More details are shown in Table 2. For instance, C20 has 17 features and 138 test cases are used to test these features. Each test case tc_i has a successful rate for execution ($SucR_{tc_i}$), an average execution time (AET_{tc_i}) and an execution frequency (EF_{tc_i}) (Section 3.2). In general, for *Saturn*, each feature is associated with 5–10 test cases and each test case tc_i is associated with 1–5 features with $SucR_{tc_i}$ ranging from 50% to 95%, AET_{tc_i} ranging from 2 min to 60 min and EF_{tc_i} ranging from 1 time per week to 50 times per week.

Moreover, we defined 500 artificial problems to empirically evaluate whether the fitness function defined in Section 3 can address our test minimization problem even with a large number of features and test cases. With this objective in mind, we first created a feature repository including 1200 features and a test case repository of 60,000 test cases. For each test case tc_i, three key attributes are assigned randomly (inspired by our industrial case study but with expansion for generality), i.e., $SucR_{tc_i}$ ranges from 0% to 100%, AET_{tc_i} ranges from 1 min to 100 min and EF_{tc_i} ranges from 1 time to 100 times per week. Moreover, we created artificial problems with the increasing number of features and test cases, i.e., we used a range of 10–1000 with an increment of 10 for features number and each feature can be associated with test cases ranging from 5 to 45 with an increment of 10. So that 100 × 5 = 500 artificial problems were obtained in this way. Based on that, each artificial problem consists of a

set of features associated with a test suite including a set of test cases and the goal is to minimize the test suite using the selected multi-objective algorithms.

5. Empirical evaluation

This section first presents experiment design including: (1) research questions required to be addressed, (2) selected criteria for the algorithms and parameter settings, and (3) evaluation mechanisms for all the search algorithms. Moreover, we also describe relevant statistical tests used for our experiments followed by experiment execution.

5.1. Experiments design

The goal of our experiments is to evaluate the fitness function in conjunction with the ten search algorithms (i.e., WGBA, WBGA-MO, RWGA, NSGA-II, MOCell, SPEA2, PAES, SMPSO, CellDE and RS) in terms of addressing the optimization problem: minimizing the test suite for testing a product with high *TMP*, *FPC*, *FDC* and *AEF*, and *OET* within allocated time budget.

5.1.1. Research questions

We will answer the following research questions with our experiments:

1. **RQ1**: Are the selected multi-objective search algorithms cost/effective to solve our test minimization problem?
2. **RQ2**: Are the selected multi-objective search algorithms cost/effective as compared to RS?
3. **RQ3**: Which one achieves the best performance among the selected multi-objective search algorithms?
4. **RQ4**: How does the increment of the number of features and test cases impact the performance of the selected multi-objective search algorithms?

5.1.2. Selected criteria for the algorithms and parameter settings

In our experiments, we first compared three weight-based multi-objective GAs and RS, i.e., WGBA with two set of fixed weights based on the domain knowledge and expertise WBGA_W$_1$ (W_1 = (0.2, 0.2, 0.2, 0.2, 0.2)), WBGA_W$_2$ (W_2 = (0.1, 0.2, 0.2, 0.4, 0.1)), WBGA-MO and RWGA. For all of them, we used a standard one-point crossover with a rate of 0.9 and mutation of a variable is done with the standard probability $1/n$, where n is the number of variables (defaulted standard parameter settings from jMetal (Durillo and Nebro, 2011)). Meanwhile, the size of population and maximum number of fitness evaluation are set as 100 and 25,000, respectively (all these standard settings performed well in our previous work (Wang et al., 2013c)). Finally, RS was used as the comparison baseline to assess the difficulty of the addressed minimization problems (Ali et al., 2010).

Moreover, we compared the best weight-based GA with the other six selected search algorithms, i.e., NSGA-II, MOCell, SPEA2, PAES, SMPSO, CellDE (RS is still used as a baseline for assessing the difficulty of the problems (Ali et al., 2010)). For all the other algorithms, we also used the default suggested parameter settings from jMetal (Durillo and Nebro, 2011) as shown in Table 3. Notice that different settings may lead to different performance for genetic algorithms, but standard settings usually perform well (Arcuri and Briand, 2011).

Table 3
Parameterization for the selected search algorithms (n is the number of variables).

Parameterization used in NSGA-II and SPEA2	
Population Size	100
Selection of Parents	Binary tournament + binary tournament
Recombination	Simulated binary, crossover rate = 0.9
Mutation	Polynomial, mutation rate = 1.0/n
Parameterization used in MOCell	
Population Size	100
Neighborhood	1-hop neighbors (8 surrounding solutions)
Selection of Parents	Binary tournament + binary tournament
Recombination	Simulated binary, crossover rate = 0.9
Mutation	Polynomial, mutation rate = 1.0/n
Archive Size	100
Parameterization used in PAES	
Mutation	Polynomial, mutation rate = 1.0/n
Archive Size	100
Parameterization used in SMPSO	
Swarm Size	100
Mutation	Polynomial, mutation rate = 1.0/n
Archive Size	100
Parameterization used in CellDE	
Population Size	100
Neighborhood	1-hop neighbors (8 surrounding solutions)
Selection of Parents	Binary tournament + binary tournament
Recombination	Differential evolution, crossover rate = 0.9
Archive Size	100

5.1.3. Evaluation mechanisms

To address RQ1, a set of thresholds for *TMP*, *FPC*, *FDC* and *OET* are selected showing the minimum acceptable values for a particular context. Notice that these thresholds are set through the domain analysis, discussion with test engineers and a test manager at Cisco, and test execution history. Meanwhile, there is no specific threshold provided for *AEF* since the test engineers think that it is not meaningful for them to propose such threshold. In our context, these thresholds values are: *TMP* := 0.8; *FPC* := 0.8; *FDC* := 0.85 and *OET* := 2 (h). Notice the purpose of these pre-defined thresholds is to assess the performance of the selected search algorithms and the obtained solutions are independent of the availability of these threshold values. In other contexts, experts can provide such thresholds if required.

As discussed in Section 2, NSGA-II, MOCell, SPEA2, PAES, SMPSO, CellDE may produce more than one solutions. Our evaluation mechanism is to choose the best solution with the highest value of objective functions for *TMP*, *FPC*, *FDC* and *OET* and compare the values of objective functions with provided thresholds separately. For instance, for *TMP* using NSGA-II, the highest value of objective function for the best solution is first chosen from the obtained ones. The value of the objective function is then compared with the threshold for *TMP*. Notice that for each objective, the chosen solution may be different so that it is essential to seek a mechanism to evaluate the overall performance of obtained solutions by the algorithms considering all the objectives together. Our mechanism is to calculate the average value to combine the five provided thresholds and the values of objective functions for the solutions obtained by all the search algorithms. The formula for combining is shown as below and the combined value is called as Overall Fitness Value (*OFV*).

$$OFV = \frac{(nor(TMP) + nor(FPC) + nor(FDC) + (1 - nor(OET)) + nor(AEF))}{5}$$

Afterwards, we can evaluate the overall performance for all the selected search algorithms using the values of *OFV*. Such mechanism makes sense since we define the same objective functions, and use the same formula to combine the values of objective functions for all the algorithms. Notice that higher the *OFV* value is, better the performance of a specific algorithm is.

To answer RQ2, RQ3 and RQ4, the same mechanisms are adapted as above when comparing all the algorithms. Notice that for the industrial case study, we first compare and analyze the results obtained by the selected algorithms for each objective separately and then evaluate the *OFV* considering the five objectives together. As for artificial problems, we only evaluate the values of *OFV* to assess the performance and scalability for the ten algorithms. Moreover, for artificial problems, Mean Fitness Value for each problem

($MFV_{i,j}$, i refers to the number of features and j means the number of associated test cases, i.e., $10 \leq i \leq 1000$ with an increment of 10 and $5 \leq j \leq 45$ with an increment of 10) is defined to measure the mean overall fitness value for a certain number of runs nr as below (in our case, $nr = 100$). $OFV_{i,j,r}$ is the fitness value of the 25,000th generation after the rth run with i features and j associated test cases.

$$MFV_{i,j} = \frac{\sum_{r=1}^{nr} OFV_{i,j,r}}{nr}$$

Moreover, Mean Fitness Value for Features (MFV_{F_i}) is defined for RQ4 to measure how the increment of the number of features impacts the performance of the selected search algorithms as below.

$$MFV_{F_i} = \frac{\sum_{j=5,j=j+10}^{45} MFV_{i,j}}{5}$$

where $10 \leq i \leq 1000$ with an increment of 10, i.e., a set of hundred *MFV_F* values can be obtained for various numbers of features.

Similarly, Mean Fitness Value for associated Test Cases (MFV_{TC_j}) is also defined for RQ4 to measure how the increment of the number of associated test cases impacts the performance of the selected multi-objective search algorithms as follows.

$$MFV_{TC_j} = \frac{\sum_{i=10,i=i+10}^{1000} MFV_{i,j}}{100}$$

where $5 \leq i \leq 45$ with an increment of 10, i.e., a set of five *MFV_TC* values can be obtained for different number of associated test cases.

5.2. Statistical tests

To compare the obtained result and given thresholds, the Vargha and Delaney statistics, Kruskal–Wallis test and Mann–Whitney *U* test are used based on the guidelines for reporting statistical tests for randomized algorithms (Arcuri and Briand, 2011). The Vargha and Delaney statistics is used to calculate $\hat{A}_{12}$, which is a non-parametric effect size measure. In our context, $\hat{A}_{12}$ is used to compare the probability of yielding higher values for each objective function and overall fitness value (*OFV*) for two algorithms *A* and *B*. If $\hat{A}_{12}$ is 0.5, the two algorithms are equivalent. If $\hat{A}_{12}$ is greater than 0.5, *A* has higher chances to obtain better solution than *B*. To determine statistical significance of results, first the Kruskal–Wallis test together with the Bonferroni Correction is performed to determine if there are significant differences among all the algorithms (Mcdonald, 2009). Notice that the Bonferroni Correction is used for adjusting *p*-value obtained by the Kruskal–Wallis test since the Kruskal–Wallis test is used to compare multiple sets of results (samples) obtained by various search algorithms in our case. If significant differences are observed, we further perform pair-wise comparisons of the algorithms using the Mann–Whitney *U* test (Mcdonald, 2009). Therefore, when comparing the performance of the selected search algorithms (RQ2 and RQ3), we first perform the Kruskal–Wallis test to assess whether there exists significant differences for all the samples obtained by various algorithms. If significant differences are observed, the Mann–Whitney *U* test is further performed together with Vargha and Delaney statistics for pair-wise comparison of the algorithms. We choose the significance level of 0.05, i.e., there is a significant difference if *p*-value is less than 0.05. Based on the above description, we follow the following convention in the paper: An algorithm *A* has better performance than algorithm *B*, if the $\hat{A}_{12}$ value is greater than 0.5 and have significantly better performance than *B* if *p*-value is less than 0.05.

Moreover, to address RQ4, we choose the Spearman's rank correlation coefficient (ρ) to measure the relations between the *MFV* of the algorithms and different number of features and test cases (Sheskin, 2007). The value of ρ ranges from -1 to 1, i.e., there is a positive correlation if ρ is equal to 1 and a negative correlation when ρ is -1. A ρ close to 0 shows that there is no correlation between the two sets of data. Moreover, we also report significance of correlation using $Prob > |\rho|$, a value lower than 0.05 means the correlation is statistically significant.

5.3. Experiment execution

According to the guidelines in Arcuri and Briand (2011), each algorithm must be run for at least 100 times to account for random variation inherited in search algorithms. All the selected algorithms are run up to 25,000 generations each time and we collected the optimal solution including the final value of fitness function for the 25,000th generation. We ran our experiments on two PCs with Intel Core i7 2.3 GHz with 4 GB of RAM, running Microsoft Windows 7 operating system. In summary, it took 204.5 h (eight and half days) to run the algorithms for all the problems.

6. Results and analysis

In this section, we present the results and analysis of the experiments for the individual research question including two parts: (1) comparing the performance of the three weight-based GAs since all of them are based on weight theory; and (2) comparing the performance of the best weight-based GA and the other selected search algorithms, which are classified into different mechanisms to assess which one can achieve the best performance for our minimization problem. Recall that RQ1–RQ3 address questions how the selected algorithms perform in terms of cost/effectiveness for our minimization problem (as compared with RS), while RQ4 tackles the question corresponded with the impact on the performance of these algorithms with the growth of features and associated test cases (Section 5.1.1).

6.1. Results and analysis for the selected weight-based GAs

In this section, we present the results and analysis for the selected weight-based GAs (i.e., WBGA_W_1, WBGA_W_2, WBGA-MO and RWGA) using our industrial case study and 500 artificial problems. Furthermore, we also report a timing analysis, where we compare time taken by all the weight-based GAs as compared with RS.

6.1.1. Results and analysis for industrial case study

We first discuss the results and analysis for the industrial case study for addressing the proposed research questions.

6.1.1.1. Results and analysis for research question 1. Table 4 shows the results, when the performance of the selected weight-based algorithms (WBGA_W_1, WBGA_W_2, WBGA-MO and RWGA) are compared with the threshold values for each objective (i.e., *TMP*, *FPC*, *FDC*, *OET*) for our industrial case study, while Table 5 describes the results for the *OFV* (i.e., overall fitness value) obtained by the selected algorithms. In these two tables, we did one sample Mann–Whitney test since we compared the results by different algorithms with one set of fixed values for *TMP*, *FPC*, *FDC*, *OET* and *OFV*. Notice that RS is a baseline for assessing the difficulty of the addressed problems.

Based on the obtained results, we can answer RQ1 as follows.

For all the objectives, WBGA-MO and RWGA have higher probability to obtain better results when compared with the given thresholds, i.e., they have higher probability to be used when required since all of the $\hat{A}_{12}$ values are greater than 0.5 for *TMP*, *FPC*

Table 4
Results for comparing the weight-based algorithms with the given thresholds for each objective.*

Products	Algorithms	Objectives with the given thresholds							
		TMP: 0.8		*FPC*: 0.8		*FDC*: 0.85		*OET*: 2 h	
		A	*p*	*A*	*p*	*A*	*p*	*A*	*p*
C20	WBGA_W$_1$	0.45	0.12	0.37	0.08	0.39	0.19	0.45	0.16
	WBGA_W$_2$	0.55	0.10	0.44	0.17	0.56	0.36	0.38	0.09
	WGBA-MO	0.61	0.17	0.67	0.13	0.54	0.32	0.39	0.08
	RWGA	0.69	0.19	0.76	0.19	0.75	0.10	0.32	**0.03**
	RS	0.22	**0.01**	0.34	**0.02**	0.47	**0.05**	0.55	0.23
C40	WBGA_W$_1$	0.55	0.40	0.44	0.18	0.41	0.16	0.39	0.08
	WBGA_W$_2$	0.68	**0.04**	0.39	0.20	0.56	0.08	0.42	0.19
	WBGA-MO	0.57	0.46	0.63	0.11	0.61	0.10	0.38	0.07
	RWGA	0.68	0.08	0.65	0.10	0.73	**0.05**	0.34	0.06
	RS	0.35	**0.02**	0.23	**0.01**	0.44	0.11	0.65	**0.03**
C60	WBGA_W$_1$	0.61	0.10	0.48	0.38	0.32	**0.04**	0.43	0.33
	WBGA_W$_2$	0.59	0.37	0.52	0.27	0.57	0.06	0.39	0.25
	WBGA-MO	0.56	0.22	0.64	0.09	0.66	0.07	0.32	0.08
	RWGA	0.67	0.13	0.71	0.17	0.78	**0.05**	0.39	0.15
	RS	0.39	**0.01**	0.32	**0.02**	0.35	**0.02**	0.71	**0.01**
C90	WBGA_W$_1$	0.64	0.21	0.30	**0.04**	0.27	**0.03**	0.41	0.27
	WBGA_W$_2$	0.66	0.18	0.53	0.12	0.37	0.10	0.43	0.35
	WGBA-MO	0.67	0.19	0.58	0.19	0.64	0.12	0.39	0.10
	RWGA	0.70	0.12	0.64	0.17	0.69	0.09	0.31	0.06
	RS	0.22	**0.02**	0.34	**0.01**	0.49	0.22	0.67	**0.02**

* *A*: $\hat{A}_{12}$, *p*: *p*-value. All *p*-values less than 0.05 are identified as bold.

and *FDC* and all of the $\hat{A}_{12}$ values are less than 0.5 for *OET*. Moreover, there are no significant differences (all *p*-values are greater than 0.05) when comparing with the given thresholds as shown in Table 4.

For WBGA_W$_1$ and WBGA_W$_2$, the results do not stay stable. For some products and objectives (e.g., for *TMP* in *C40* product), WBGA_W$_1$ and WBGA_W$_2$ have more chance to achieve better results than the given thresholds since the $\hat{A}_{12}$ values are greater than 0.5 (Table 4). But for the other products and objectives (e.g., for *FDC* in *C90* product), WBGA_W$_1$ and WBGA_W$_2$ have less chance to obtain the better results as compared with the given thresholds since the $\hat{A}_{12}$ values are much less than 0.5 (Table 4). Since the only difference between WBGA_W$_1$ and WBGA_W$_2$ is the sets of weights, different weights can result in total different results when using WBGA. So providing an appropriate set of weights before using WBGA is essential to obtain the expected results.

Based on the results, we can see that RS always has less probability to be used in practice when compared with the given thresholds since all the $\hat{A}_{12}$ values are less than 0.5 for *TMP*, *FPC* and *FDC* and all of the $\hat{A}_{12}$ values are greater than 0.5 for *OET*. Moreover, there are significant differences when comparing with the given thresholds since most of the *p*-values are less than 0.05 as shown in Table 4.

When comparing the results for *OFV* considering all the objectives together, WGBA-MO and RWGA have higher probability to achieve better *OFV* than the given thresholds since all the $\hat{A}_{12}$ values are greater than 0.5 and there are almost no significant differences (most of the *p*-values are greater than 0.05) as shown in Table 5. As for WBGA (WBGA_W$_1$ and WBGA_W$_2$), the results do not stay stable, i.e., some of them have higher probability to obtain better *OFV* as compared with the thresholds while some others have less probability to achieve better *OFV* than the thresholds (Table 5). However, RS always has less probability to obtain better *OFV* than the thresholds since all the $\hat{A}_{12}$ values are greater than 0.5 and all the *p*-values are less than 0.05 (Table 5).

In summary, weight-based GAs can assist to achieve the given thresholds. However, for WBGA (i.e., WGBA_W$_1$ and WGBA_W$_2$), determining appropriate weights is challenging as it mostly dependent on domain-expertise and expert-guessing of what should be the most appropriate values, which is the main reason why WBGA with various weights can achieve different weights. But WGBA-MO and RWGA are not dependent on the pre-defined weights thus the obtained results are more stable.

6.1.1.2. Results and analysis for research question 2 and 3. For the *Saturn* products, the Kruskal–Wallis test (together with the Bonferroni Correction) was first performed for all the samples obtained by weight-based GAs and RS with respect to each objective and *OFV* (considering all the objective together, Section 5.1.3) to determine whether there are significant differences between the performances of the algorithms. We obtained the following *p*-values: 0.017 for *TMP*, 0.008 for *FPC*, 0.020 for *FDC*, 0.012 for *OET* and 0.010 for *OFV*, which shows that there exist significant differences between the performance of the weight-based GAs and RS since all the *p*-values are less than 0.05. Therefore, the Vargha and Delaney statistics and the Mann–Whitney *U* test were further performed to compare each

Table 5
Results for comparing the weight-based algorithms with the given thresholds for *OFV*.*

Pair of algorithms	C20		C40		C60		C90	
	A	*p*	*A*	*p*	*A*	*p*	*A*	*p*
WBGA_W1 vs. Thresholds	0.51	0.23	0.32	**0.04**	0.35	**0.04**	0.40	0.12
WBGA_W2 vs. Thresholds	0.52	0.18	0.41	0.12	0.59	0.14	0.47	0.19
WBGA-MO vs. Thresholds	0.56	0.11	0.55	0.15	0.62	0.09	0.56	0.16
RWGA vs. Thresholds	0.62	0.32	0.59	0.18	0.67	0.07	0.71	**0.04**
RS vs. Thresholds	0.28	**0.02**	0.33	**0.02**	0.39	**0.01**	0.27	**0.01**

* All *p*-values less than 0.05 are identified as bold.

Table 6
Results for comparing the weight-based algorithms for each objective.*

Products	Pair of algorithms	Objectives							
		TMP		*FPC*		*FDC*		*OET*	
		A	*p*	*A*	*p*	*A*	*p*	*A*	*p*
C20	WBGA_W_1 vs. WBGA_W_2	0.57	0.29	0.58	0.15	0.45	0.27	0.54	0.22
	WBGA_W_1 vs. WBGA-MO	0.35	**0.05**	0.41	0.38	0.40	0.27	0.32	0.08
	WBGA_W_1 vs. RWGA	0.22	**0.02**	0.40	0.25	0.27	0.10	0.38	0.24
	WBGA_W_1 vs. RS	0.66	**0.04**	0.58	0.17	0.69	**0.04**	0.68	**0.02**
	WBGA_W_2 vs. WBGA-MO	0.54	0.12	0.47	0.53	0.40	0.14	0.59	0.08
	WBGA_W_2 vs. RWGA	0.28	**0.02**	0.32	**0.02**	0.41	0.13	0.30	**0.02**
	WBGA_W_2 vs. RS	0.65	**0.02**	0.66	**0.02**	0.57	0.14	0.64	**0.03**
	WBGA-MO vs. RWGA	0.30	**0.01**	0.32	**0.02**	0.34	**0.02**	0.38	0.06
	WBGA-MO vs. RS	0.67	**0.02**	0.70	**0.01**	0.58	0.09	0.69	**0.02**
	RWGA vs. RS	0.61	0.06	0.73	**0.01**	0.59	0.10	0.79	**0.01**
C40	WBGA_W_1 vs. WBGA_W_2	0.52	0.44	0.47	0.16	0.51	0.45	0.56	0.30
	WBGA_W_1 vs. WBGA-MO	0.45	0.38	0.35	0.29	0.38	0.22	0.33	0.14
	WBGA_W_1 vs. RWGA	0.18	**0.01**	0.26	0.09	0.41	0.37	0.32	0.11
	WBGA_W_1 vs. RS	0.67	**0.03**	0.60	0.18	0.69	**0.03**	0.70	**0.01**
	WBGA_W_2 vs. WBGA-MO	0.48	0.21	0.42	0.23	0.52	0.37	0.43	0.12
	WBGA_W_2 vs. RWGA	0.31	**0.03**	0.40	0.09	0.29	**0.02**	0.37	0.10
	WBGA_W_2 vs. RS	0.61	**0.03**	0.59	**0.04**	0.65	**0.03**	0.67	**0.02**
	WBGA-MO vs. RWGA	0.33	**0.02**	0.29	**0.02**	0.37	**0.03**	0.39	0.08
	WBGA-MO vs. RS	0.66	**0.03**	0.69	**0.02**	0.60	**0.04**	0.65	**0.03**
	RWGA vs. RS	0.82	**0.01**	0.76	**0.02**	0.69	**0.02**	0.59	**0.01**
C60	WBGA_W_1 vs. WBGA_W_2	0.53	0.43	0.58	0.15	0.46	0.21	0.47	0.34
	WBGA_W_1 vs. WBGA-MO	0.52	0.46	0.36	0.19	0.29	**0.02**	0.34	0.11
	WBGA_W_1 vs. RWGA	0.35	**0.04**	0.17	**0.01**	0.25	0.07	0.34	0.15
	WBGA_W_1 vs. RS	0.59	0.27	0.65	**0.04**	0.68	**0.04**	0.56	0.29
	WBGA_W_2 vs. WBGA-MO	0.57	0.09	0.43	0.22	0.47	0.39	0.54	0.27
	WBGA_W_2 vs. RWGA	0.30	**0.03**	0.21	**0.01**	0.29	**0.01**	0.32	**0.02**
	WBGA_W_2 vs. RS	0.60	**0.04**	0.59	**0.04**	0.61	**0.03**	0.56	0.11
	WBGA-MO vs. RWGA	0.39	0.14	0.29	**0.02**	0.30	**0.03**	0.32	**0.03**
	WBGA-MO vs. RS	0.60	**0.03**	0.68	**0.02**	0.55	0.16	0.64	**0.02**
	RWGA vs. RS	0.59	**0.03**	0.62	**0.02**	0.74	**0.01**	0.66	**0.03**
C90	WBGA_W_1 vs. WBGA_W_2	0.48	0.39	0.57	0.16	0.55	0.21	0.49	0.44
	WBGA_W_1 vs. WBGA-MO	0.40	0.15	0.31	0.07	0.41	0.17	0.26	**0.04**
	WBGA_W_1 vs. RWGA	0.26	**0.03**	0.36	0.12	0.38	0.24	0.19	**0.01**
	WBGA_W_1 vs. RS	0.57	0.10	0.67	**0.03**	0.59	0.08	0.65	**0.04**
	WBGA_W_2 vs. WBGA-MO	0.44	0.20	0.46	0.31	0.48	0.23	0.42	0.10
	WBGA_W_2 vs. RWGA	0.37	**0.05**	0.40	0.13	0.31	**0.03**	0.29	**0.01**
	WBGA_W_2 vs. RS	0.62	**0.03**	0.65	**0.02**	0.68	**0.02**	0.60	**0.03**
	WBGA-MO vs. RWGA	0.36	**0.03**	0.39	**0.05**	0.30	**0.02**	0.35	**0.03**
	WBGA-MO vs. RS	0.64	**0.03**	0.67	**0.02**	0.70	**0.01**	0.60	**0.03**
	RWGA vs. RS	0.69	**0.01**	0.71	**0.01**	0.77	**0.01**	0.63	**0.02**

* All *p*-values less than 0.05 are identified as bold.

pair of weight-based GAs as shown in Tables 6 and 7. In total, we have $C_2^5 = 10$ pairs to compare for the three selected search algorithms. The obtained results are first compared for each objective (Table 6) and then compared for *OFV* considering all the objectives (Table 7).

Based on the obtained results, we can answer RQ2 and RQ 3 as follows:

RQ2: For all the objectives (i.e., *TMP*, *FPC*, *FDC* and *OET*), the results show that all the selected weight-based GAs have significantly better performance than RS since all the $\hat{A}_{12}$ values for *TMP*, *FPC*, and *FDC* are greater than 0.5, and all the $\hat{A}_{12}$ values for *OET* are less than 0.5, and most of the *p*-values are less than 0.05 (Table 6). When comparing the *OFV* results considering all objectives for the algorithms and RS, we can see that all of the selected weight-based

Table 7
Results for comparing the weight-based algorithms for *OFV*.*

Pair of algorithms	C20		C40		C60		C90	
	A	*p*	*A*	*p*	*A*	*p*	*A*	*p*
WBGA_W_1 vs. WBGA_W_2	0.41	0.23	0.42	0.38	0.51	0.46	0.46	0.26
WBGA_W_1 vs. WBGA-MO	0.36	0.29	0.32	**0.05**	0.40	0.26	0.29	**0.02**
WBGA_W_1 vs. RWGA	0.18	**0.01**	0.27	**0.03**	0.21	**0.02**	0.22	**0.02**
WBGA_W_1 vs. RS	0.60	**0.04**	0.69	**0.01**	0.57	0.11	0.63	**0.03**
WBGA_W_2 vs. WBGA-MO	0.39	**0.04**	0.40	0.12	0.37	**0.04**	0.43	0.22
WBGA_W_2 vs. RWGA	0.33	**0.02**	0.30	**0.02**	0.38	**0.04**	0.31	**0.02**
WBGA_W_2 vs. RS	0.60	**0.04**	0.59	**0.04**	0.63	**0.03**	0.67	**0.03**
WBGA-MO vs. RWGA	0.39	**0.04**	0.32	**0.02**	0.37	**0.04**	0.32	**0.02**
WBGA-MO vs. RS	0.60	**0.03**	0.69	**0.02**	0.62	**0.03**	0.64	**0.03**
RWGA vs. RS	0.66	**0.01**	0.79	**0.02**	0.65	**0.02**	0.72	**0.02**

* All *p*-values less than 0.05 are identified as bold.

Table 8
Results for the Vargha and Delaney statistics-artificial problems (without/with Whitney *U* test).

RQ	Pair of algorithms (*A* vs. *B*)	$A > B$	$A < B$	$A = B$
RQ2	WBGA_W$_1$ vs. RS	367/354	133/102	0/54
	WBGA_W$_2$ vs. RS	392/371	108/96	0/33
	WBGA_MO vs. RS	445/426	55/45	0/29
	RWGA vs. RS	489/480	11/0	0/20
RQ3	WBGA_W$_1$ vs. WBGA_W$_2$	226/202	274/264	0/34
	WBGA_W$_1$ vs. WBGA-MO	198/170	302/287	0/43
	WBGA_W$_1$ vs. RWGA	119/93	381/371	0/36
	WBGA_W$_2$ vs. WBGA-MO	211/198	289/264	0/38
	WBGA_W$_2$ vs. RWGA	134/122	366/354	0/24
	WBGA-MO vs. RWGA	192/174	308/295	0/31

GAs have significantly better performance than RS since all the $\hat{A}_{12}$ values are greater than 0.5 and all the *p*-values are less than 0.05 (Table 7).

RQ3: For all the objectives (i.e., *TMP*, *FPC*, *FDC* and *OET*), the results show that RWGA significantly outperforms WBGA_W$_1$, WBGA_W$_2$ and WBGA-MO since all the $\hat{A}_{12}$ values for *TMP*, *FPC*, and *FDC* are greater than 0.5, and all the $\hat{A}_{12}$ values for *OET* are less than 0.5, and most of the *p*-values are less than 0.05 (Table 6). Similarly, when comparing *OFV*, the performance of RWGA is also significantly better than the other weight-based GAs since all the $\hat{A}_{12}$ values are greater than 0.5 and all the *p*-values are less than 0.05 (Table 7).

In general, we can conclude that RWGA achieves the best performance among the selected weight-based GAs for all the objectives and overall fitness value (*OFV*) when considering all the objectives together.

6.1.2. Results and analysis for artificial problems

To answer RQ2 and RQ3, we first conducted the Kruskal–Wallis test (together with the Bonferroni Correction) for all the obtained results and we obtained a *p*-value of 0.014 (less than 0.05) showing that there are significant differences between the results obtained by the selected weight-based GAs and RS. Moreover, we further compared the selected weight-based algorithms with RS and then compared each pair of them based on mean fitness value (*MFV*) obtained after 25,000 generations for each algorithm and each of the problems. Recall that each problem was repeated for 100 times to account for random variation (i.e., *MFV* is a mean for 500 values of *OFV* after 100 runs). Notice that we answer RQ4 in terms of scalablity for all the ten search algorithms in Section 6.2.2.2.

Table 8 summarizes the results for Vargha and Delaney statistics without/with Whitney *U* test for the 500 artificial problems. Two numbers are shown in each cell of the table splited by a slash. The first number in the column $A > B$ shows the number of problems out of 500 for which an algorithm *A* has better performance than *B* ($\hat{A}_{12} > 0.5$), $A < B$ means vice versa ($\hat{A}_{12} < 0.5$), and $A = B$ means the number of problems for which *A* has equivalent performance as *B* ($\hat{A}_{12} = 0.5$). The second number after "/" in the column $A > B$ means the number of problems out of 500 for which an algorithm *A* has significantly better performance than *B* ($\hat{A}_{12} > 0.5$ and $p < 0.05$), $A < B$ means vice versa ($\hat{A}_{12} < 0.5$ and $p < 0.05$), and $A = B$ means the number of problems for which there are no siginificant differences in performance between *A* and *B* ($p \geq 0.05$).

6.1.2.1. Results and analysis for research question 2 and 3. To answer RQ2 and RQ3, we compared WBGA_W$_1$, WBGA_W$_2$, WBGA-MO, RWGA and RS based on the mean fitness values for each problem ($MFV_{i,j}$, where $10 \leq i \leq 1000$ with an increment of 10 and $5 \leq j \leq 45$ with an increment of 10, i.e., there are 500 *MFV* values for the 500 artificial problems).

Table 9
Average running time for the selected weight-based search algorithms.

Algorithms	WBGA_W$_1$	WBGA_W$_2$	WBGA-MO	RWGA	RS
Time (s)	2.01	2.11	1.94	1.37	1.13

RQ2: All the selected weight-based GAs significantly outperformed RS for finding optimal solutions in most of the problems. For instance, WBGA-MO has better performance than RS for 445 problems and the performance is significantly better for 426 problems. WBGA-MO has worse performance than RS for 55 problems (45 of them were significantly worse). There were no significant differences between WBGA-MO and RS for 29 problems (Table 8).

RQ3: RWGA achieved the best performance among all the other algorithms in most of the problems (the performance of RWGA is better than the other algorithms for 351.7 problems on average $((381 + 366 + 308)/3 = 351.7)$ and significantly better for 340 problems on average $((371 + 354 + 295)/3 = 340)$). WBGA_W$_1$ and WBGA_W$_2$ have almost equivalent performance, whose performance stays at the second level. Moreover, WBGA-MO achieved better performance than WBGA_W$_1$ and WBGA_W$_2$ (WBGA-MO outperformed WBGA_W$_1$ and WBGA_W$_2$ for 302 and 289 problems respectively and significantly outperformed for 287 and 264 problems respectively).

6.1.3. Timing analysis for weight-based GAs

In addition, we report the average time taken by the three weight-based GAs per run (i.e., WBGA, WBGA-MO and RWGA) as compared with RS (shown in Table 9). In addition, the Kruskal–Wallis test was conducted to evaluate whether there are significant differences among the running time of the weight-based GAs as compared with RS (100 run in our case). The *p*-value obtained by the Kruskal–Wallis test is 0.39, which shows that there are no significant differences among the running time of the weight-based GAs and RS. Based on the results, we can conclude that the selected weight-based GAs took similar running time as compared with RS in terms of finding an optimal solution for our test minimization problem.

6.2. Results and analysis for the best weight-based GA (RWGA) and the other selected search algorithms

This section presents the results and related analysis for our industrial case study and the designed artificial problems using the best weight-based GA (RWGA) and the other selected algorithms.

6.2.1. Results and analysis for industrial case study

In this section, we summarize the results in the following sections for the industrial case study.

6.2.1.1. Analysis for research question 1. We did one sample Mann–Whitney test since we compared the results by different algorithms with one set of fixed values for *TMP*, *FPC*, *FDC*, *OET* and *OFV* (i.e., overall fitness value). Tables 10 and 11 show that the performance of algorithms as compared with the threshold values for each objective (i.e., *TMP*, *FPC*, *FDC*, *OET* and *OFV*). Based on the results, we answer RQ1 as follows.

For *TMP*, all the selected algorithms except RS have higher probability to achieve better results than the given threshold for *TMP* since most of the $\hat{A}_{12}$ values are greater than 0.5 and there are no significant differences between the results and the threshold (all the *p*-values are greater than 0.05). However, all the values of $\hat{A}_{12}$ obtained by RS are less than 0.5 and all the *p*-values are less than 0.05, i.e., RS has

Table 10
Results for comparing the eight algorithms with the given thresholds for each objective.*

Products	Algorithms	Objectives with the given thresholds							
		TMP: 0.8		*FPC*: 0.8		*FDC*: 0.85		*OET*: 2 h	
		A	*p*	*A*	*p*	*A*	*p*	*A*	*p*
C20	RWGA	0.69	0.19	0.76	0.19	0.75	0.10	0.32	**0.03**
	NSGA-II	0.72	0.23	0.63	0.33	0.73	0.21	0.29	**0.01**
	MOCell	0.65	0.32	0.61	0.34	0.57	0.17	0.34	**0.03**
	SPEA2	0.59	0.18	0.38	**0.04**	0.39	0.21	0.25	**0.02**
	PAES	0.68	0.24	0.54	0.27	0.54	0.18	0.43	0.12
	SMPSO	0.57	0.32	0.62	0.31	0.63	0.35	0.39	0.08
	CellDE	0.63	0.35	0.68	0.34	0.57	0.29	0.28	**0.02**
	RS	0.22	**0.01**	0.34	**0.02**	0.47	**0.05**	0.55	0.23
C40	RWGA	0.68	0.08	0.65	0.10	0.73	**0.05**	0.34	0.06
	NSGA-II	0.55	0.22	0.65	0.18	0.68	0.32	0.34	**0.02**
	MOCell	0.72	0.31	0.58	0.41	0.61	0.24	0.27	**0.04**
	SPEA2	0.57	0.25	0.31	**0.04**	0.33	**0.04**	0.25	**0.02**
	PAES	0.61	0.34	0.62	0.19	0.54	0.19	0.42	0.17
	SMPSO	0.65	0.17	0.57	0.23	0.64	0.27	0.38	0.07
	CellDE	0.72	0.19	0.61	0.31	0.59	0.34	0.26	**0.02**
	RS	0.35	**0.02**	0.23	**0.01**	0.44	0.11	0.65	**0.03**
C60	RWGA	0.67	0.13	0.71	0.17	0.78	**0.05**	0.39	0.15
	NSGA-II	0.61	0.32	0.55	0.23	0.62	0.11	0.48	0.21
	MOCell	0.54	0.22	0.63	0.18	0.57	0.24	0.35	**0.04**
	SPEA2	0.65	0.34	0.49	0.17	0.31	**0.03**	0.37	**0.03**
	PAES	0.47	0.41	0.57	0.23	0.62	0.18	0.42	0.11
	SMPSO	0.49	0.25	0.63	0.29	0.55	0.23	0.30	**0.04**
	CellDE	0.54	0.18	0.59	0.32	0.59	0.31	0.29	**0.03**
	RS	0.39	**0.01**	0.32	**0.02**	0.35	**0.02**	0.71	**0.01**
C90	RWGA	0.70	0.12	0.64	0.17	0.69	0.09	0.31	0.06
	NSGA-II	0.58	0.25	0.69	0.17	0.57	0.34	0.28	**0.03**
	MOCell	0.61	0.27	0.54	0.21	0.62	0.17	0.34	0.07
	SPEA2	0.48	0.14	0.47	0.09	0.43	0.08	0.24	**0.02**
	PAES	0.54	0.11	0.49	0.15	0.54	0.13	0.42	0.13
	SMPSO	0.63	0.35	0.55	0.23	0.65	0.25	0.31	**0.04**
	CellDE	0.68	0.29	0.59	0.17	0.71	0.23	0.23	**0.02**
	RS	0.22	**0.02**	0.34	**0.01**	0.49	0.22	0.67	**0.02**

* *A*: $\hat{A}_{12}$, *p*: *p*-value. All *p*-values less than 0.05 are identified as bold.

significantly less probability to obtain better results than the given threshold (Table 10).

As for *FPC* and *FDC*, all the selected algorithms except SPEA2 and RS have higher chances to obtain better results than the given thresholds when required since almost all the values of $\hat{A}_{12}$ are greater than 0.5 and there are no significant differences (all the *p*-values are greater than 0.05). For SPEA2, all the $\hat{A}_{12}$ values are less than 0.5 and half of the *p*-values are less than 0.05, i.e., SPEA2 has significantly less probability to yield better results than the given thresholds. RS also has significantly less chance to be used for achieving an acceptable level of *FPC* and *FDC* since all the values of $\hat{A}_{12}$ are less than 0.5 and most of the *p*-values are less than 0.05 (Table 10).

For *OET*, most of the $\hat{A}_{12}$ values achieved by the selected algorithms except RS are less than 0.5 and more than half of the *p*-values are less than 0.05, i.e., they have significantly higher chances to meet the time budget within the given threshold (2 h). Moreover, all the $\hat{A}_{12}$ values obtained by RS are greater than 0.5 and most of the *p*-values are less than 0.05, i.e., the whole execution time achieved by RS has higher chance to be significantly more than 2 h (Table 10).

When comparing results for *OFV* considering all the objectives, the selected algorithms except SPEA2 and RS have higher probability to obtain better *OFV* when compared with the given thresholds since all the $\hat{A}_{12}$ values are greater than 0.5 and there are no significant differences (all the *p*-values are greater than 0.05). As for SPEA2, the obtained *OFV* has significantly less chance to be equivalent as the given thresholds (most of the $\hat{A}_{12}$ values are less than 0.5 and half of the *p*-values are less than 0.05). For RS, the obtained *OFV* has also higher chance to be significantly less than the thresholds since all the

Table 11
Results for comparing the eight algorithms with the given thresholds for *OFV*.*

Pair of algorithms	C20		C40		C60		C90	
	A	*p*	*A*	*p*	*A*	*p*	*A*	*p*
RWGA vs. Thresholds	0.62	0.32	0.59	0.18	0.67	0.07	0.71	**0.04**
NSGA-II vs. Thresholds	0.71	0.28	0.62	0.21	0.54	0.12	0.66	0.19
MOCell vs. Thresholds	0.58	0.24	0.62	0.19	0.64	0.21	0.59	0.16
SPEA2 vs. Thresholds	0.38	0.11	0.31	**0.04**	0.29	**0.03**	0.51	0.13
PAES vs. Thresholds	0.55	0.23	0.58	0.16	0.61	0.27	0.60	0.33
SMPSO vs. Thresholds	0.65	0.27	0.72	0.34	0.69	0.27	0.74	0.25
CellDE vs. Thresholds	0.59	0.18	0.63	0.22	0.74	0.39	0.69	0.35
RS vs. Thresholds	0.28	**0.02**	0.33	**0.02**	0.39	**0.01**	0.27	**0.01**

* All *p*-values less than 0.05 are identified as bold.

Table 12
Number of solutions obtained by each search algorithm.

Algorithms	WBGA_W1	WBGA_W2	WBGA-MO	RWGA	NSGA-II	MOCell
Solution No.	1	1	1	1	11	15
SPEA2	PAES	SMPSO	CellDE	RS		
8	5	19	20	0		

$\hat{A}_{12}$ values are less than 0.5 and all the *p*-values are less than 0.05 (Table 11).

We also report the number of solutions that satisfy all the given thresholds (i.e., 0.8 for *TMP*, 0.8 for *FPC*, 0.85 for *FDC* and 2 h for *OET*) obtained by each selected algorithm as shown in Table 12. Based on the results, we observe that weight-based GAs can manage to find one solution which balances all the objectives together (i.e., maximum of the *OFV* value) while the Pareto-based algorithms can obtain a set of non-dominated solutions, e.g., eleven non-dominiated solutions are obtained by using NSGA-II. Notice RS can not manage to find a solution which can satisfy the required thresholds for all the objectives.

In general, the selected algorithms except SPEA2 and RS can achieve the given thresholds for each objective well and the overall performance still works well in terms of *OFV*.

6.2.1.2. Analysis for research question 2 and 3. Similarly as Section 6.1.1.2, for the *Saturn* products, all the *p*-values obtained by the Kruskal–Wallis test (together with the Bonferroni Correction) are less than 0.05 for each objective and *OFV* (0.009 for *TMP*, 0.031 for *FPC*, 0.015 for *FDC*, 0.024 for *OET* and 0.016 for *OFV*), showing that there are significant differences between the performance of the selected search algorithms for each objective and *OFV*. Therefore, we further performed the Vargha and Delaney statistics and the Mann–Whitney *U* test to compare each pair of the eight search algorithms as shown in Tables 13 and 14. In total, we have $C_2^8 = 28$ pairs to compare for the eight selected search algorithms. The obtained results are first compared for each objective and then compared for *OFV* considering all the objectives (shown as Tables 13 and 14, respectively). Based on the results, we answer the RQ2 and RQ3 as follows.

RQ2: For *TMP*, *FPC*, *FDC* and *OET*, we first compared the performance of the selected algorithms with RS. The results show that all the selected search algorithms have significantly better performance than RS for *TMP* and *OET* since all the $\hat{A}_{12}$ values for *TMP* are greater than 0.5, and all the $\hat{A}_{12}$ values for *OET* are less than 0.5, and most of the *p*-values are less than 0.05. For *FPC* and *FDC*, all the selected algorithms except SPEA2 achieve significantly better performance than RS since almost all the $\hat{A}_{12}$ values are greater than 0.5 and most of the *p*-values are less than 0.05. SPEA2 has almost equivalent performance as RS for *FPC* and *FDC* since most of the $\hat{A}_{12}$ values are close to 0.5 and the *p*-values are greater than 0.05 (Table 13).

When comparing the *OFV* results considering all objectives for the algorithms and RS, we can see that six of them (i.e., RWGA, NSGA-II, MOCell, PAES, SMPSO and CellDE) have significantly better performance than RS since all the $\hat{A}_{12}$ values are greater than 0.5 and all the *p*-values are less than 0.05. SPEA2 has almost equivalent performance as RS since most of the $\hat{A}_{12}$ values are closer to 0.5 and the *p*-values are greater than 0.05 (Table 14).

RQ3: For *TMP*, CellDE achieved the best performance than the other algorithms since most of the $\hat{A}_{12}$ values are greater than 0.5 and the *p*-values are less than 0.05. Similarly, MOCell, NSGA-II and SPEA2 have the best performance for *FPC*, *FDC* and *OET* among all the algorithms, respectively (Table 13).

When comparing the *OFV* results considering all objectives together, RWGA performs significantly better among all the search algorithms since all the $\hat{A}_{12}$ values are greater than 0.5 and most of *p*-values are less than 0.05. Moreover, SPEA2 has significantly worse performance than the other six search algorithms since most of the $\hat{A}_{12}$ values are less than 0.5 and *p*-values are greater than 0.05 (Table 14).

In general, we can conclude that different algorithms can achieve the best performance for different objectives (e.g., CellDE for the objective *TMP*) but RWGA has significantly better performance than the other search algorithms when considering all the objectives together in general.

6.2.2. Results and analysis for artificial problems

In this section, we summarize the results as the following sections for the designed 500 artificial problems.

6.2.2.1. Results and analysis for research question 2 and 3. To answer RQ2 and RQ3, the Kruskal–Wallis test (together with the Bonferroni Correction) was first performed for the results of the eight search algorithms and we obtained a *p*-value of 0.024 (less than 0.05) showing that there exist significant differences between the performances of the selected algorithms. Therefore, we further compared the algorithms (i.e., RWGA, NSGA-II, MOCell, SPEA2, PAES, SMPSO, CellDE) with RS and then compared each pair of them based on mean fitness value (*MFV*) achieved after 25,000 generations for each algorithm and each of the 500 problems. Recall that each problem was repeated for 100 times to account for random variation, i.e., *MFV* is a mean for 100 values of *OFV* after 100 runs (Section 5.1.3). Table 14 summarizes the results for Vargha and Delaney statistics without or with Whitney *U* test for the 500 artificial problems (similarly as Table 13).

RQ2: RWGA, NSGA-II, MOCell, PAES, SMPSO, and CellDE achieved significantly better performance than RS for finding optimal solutions for our minimization problem in most of the problems. For instance, RWGA has better performance than RS for 489 problems and the performance is significantly better for 480 problems. RWGA has worse performance than RS for 11 problems (none of them was significantly worse). There were no significant differences between RWGA and RS for 20 problems. The performance of SPEA2 is close to RS (SPEA2 is better than RS for 298 problems and significantly better for 251 problems. SPEA2 is worse than RS for 202 problems and significantly worse for 179 problems. Meanwhile, there is no significant difference between them for 70 problems) (Table 15).

RQ3: RWGA achieved the best performance among all the other six algorithms in most of the problems, i.e., the performance of RWGA is better than the other six algorithms for average 418.8 problems ($(432 + 403 + 464 + 381 + 453 + 380)/6 = 418.8$) and significantly better for 401.7 problems on average ($(405 + 397 + 451 + 362 + 438 + 357)/6 = 401.7$). NSGA-II, MOCell, PAES, SMPSO, and CellDE have almost equivalent performance, whose performance stay at the second level. Finally, the performance of SPEA2 is worse than the above-mentioned algorithms, i.e., SPEA2 achieved the worst performance in the 500 artificial problems (SPEA2 is worse than the other search algorithms for 377.8 problems and significantly worse for 354.7 problems on average) (Table 15).

6.2.2.2. Results and analysis for research question 4. To answer RQ4, we calculated Spearman's rank correlation bettween mean fitness values for each algorithm with increasing number of features (MFV_F_i, where $10 \leq i \leq 1000$ with an increment of 10, Section 5.1.3) and associated test cases (MFV_TC_j, where $5 \leq j \leq 45$ with an increment of 10, Section 5.1.3). In this case, a fixed number of features is used to represent a product and each feature is associated with different number of test cases. So we have $100 \times 5 = 500$ artificial problems

Table 13
Results for comparing the eight algorithms for each objective.*

Pair of algorithms	Objectives							
	TMP: 0.8		*FPC*: 0.8		*FDC*: 0.85		*OET*: 2 h	
	A	*p*	*A*	*p*	*A*	*p*	*A*	*p*
(1) Product C20								
RWGA vs. NSGA-II	0.43	0.11	0.34	**0.04**	0.40	0.08	0.53	0.25
RWGA vs. MOCell	0.51	0.12	0.37	**0.04**	0.36	**0.04**	0.55	0.19
RWGA vs. SPEA2	0.48	0.11	0.57	0.07	0.29	**0.03**	0.37	**0.03**
RWGA vs. PAES	0.47	0.21	0.45	0.18	0.54	0.25	0.51	0.22
RWGA vs. SMPSO	0.42	0.24	0.52	0.13	0.47	0.28	0.53	0.17
RWGA vs. CellDE	0.36	**0.04**	0.43	0.08	0.52	0.14	0.55	0.21
RWGA vs. RS	0.61	0.06	0.73	**0.01**	0.59	0.10	0.79	**0.01**
NSGA-II vs. MoCell	0.51	0.17	0.41	0.13	0.67	**0.02**	0.48	0.24
NSGA-II vs. SPEA2	0.46	0.23	0.75	**0.01**	0.70	**0.02**	0.37	**0.02**
NSGA-II vs. PAES	0.54	0.28	0.53	0.24	0.65	**0.02**	0.53	0.25
NSGA-II vs. SMPSO	0.46	0.17	0.55	0.18	0.68	**0.02**	0.48	0.17
NSGA-II vs. CellDE	0.49	0.35	0.47	0.23	0.64	**0.02**	0.54	0.28
NSGA-II vs. RS	0.72	**0.01**	0.82	**0.02**	0.73	**0.01**	0.65	**0.01**
MOCell vs. SPEA2	0.58	0.17	0.75	**0.01**	0.68	**0.02**	0.35	**0.01**
MOCell vs. PAES	0.56	0.14	0.68	**0.02**	0.54	0.19	0.52	0.14
MOCell vs. SMPSO	0.54	0.27	0.65	**0.02**	0.49	0.25	0.46	0.21
MOCell vs. CellDE	0.31	**0.03**	0.66	**0.02**	0.51	0.31	0.43	0.27
MOCell vs. RS	0.69	**0.04**	0.79	**0.01**	0.69	**0.02**	0.67	**0.02**
SPEA2 vs. PAES	0.51	0.34	0.34	**0.01**	0.36	**0.01**	0.71	**0.01**
SPEA2 vs. SMPSO	0.45	0.22	0.29	**0.01**	0.31	**0.01**	0.67	**0.02**
SPEA2 vs. CellDE	0.28	**0.01**	0.31	**0.01**	0.28	**0.01**	0.72	**0.01**
SPEA2 vs. RS	0.69	**0.02**	0.52	0.21	0.54	0.17	0.75	**0.01**
PAES vs. SMPSO	0.43	0.29	0.46	0.17	0.49	0.24	0.48	0.17
PAES vs. CellDE	0.46	0.31	0.43	0.25	0.47	0.29	0.45	0.23
PAES vs. RS	0.67	**0.02**	0.63	**0.03**	0.68	**0.02**	0.67	**0.02**
SMPSO vs. CellDE	0.31	**0.01**	0.55	0.24	0.53	0.19	0.51	0.34
SMPSO vs. RS	0.65	**0.03**	0.69	**0.03**	0.72	**0.01**	0.66	**0.02**
CellDE vs. RS	0.76	**0.01**	0.67	**0.03**	0.77	**0.01**	0.63	**0.02**
(2) Product C40								
RWGA vs. NSGA-II	0.48	0.21	0.46	0.17	0.38	**0.03**	0.34	**0.02**
RWGA vs. MOCell	0.49	0.12	0.41	0.07	0.32	**0.03**	0.51	0.22
RWGA vs. SPEA2	0.52	0.23	0.52	0.18	0.31	**0.03**	0.33	**0.03**
RWGA vs. PAES	0.49	0.16	0.53	0.20	0.52	0.21	0.54	0.19
RWGA vs. SMPSO	0.47	0.19	0.48	0.27	0.45	0.29	0.49	0.23
RWGA vs. CellDE	0.32	**0.02**	0.47	0.14	0.55	0.20	0.47	0.18
RWGA vs. RS	0.82	**0.01**	0.76	**0.02**	0.69	**0.02**	0.59	**0.01**
NSGA-II vs. MoCell	0.48	0.20	0.43	0.16	0.69	**0.02**	0.51	0.17
NSGA-II vs. SPEA2	0.51	0.11	0.73	**0.01**	0.73	**0.01**	0.33	**0.01**
NSGA-II vs. PAES	0.52	0.16	0.55	0.18	0.69	**0.02**	0.48	0.21
NSGA-II vs. SMPSO	0.49	0.20	0.52	0.20	0.64	**0.03**	0.49	0.08
NSGA-II vs. CellDE	0.29	**0.02**	0.49	0.09	0.72	**0.01**	0.52	0.14
NSGA-II vs. RS	0.89	**0.02**	0.81	**0.01**	0.77	**0.01**	0.75	**0.01**
MOCell vs. SPEA2	0.54	0.19	0.77	**0.01**	0.62	0.06	0.32	**0.01**
MOCell vs. PAES	0.49	0.09	0.73	**0.01**	0.56	0.09	0.48	0.22
MOCell vs. SMPSO	0.51	0.23	0.69	**0.02**	0.53	0.18	0.49	0.32
MOCell vs. CellDE	0.29	**0.01**	0.68	**0.03**	0.54	0.27	0.47	0.18
MOCell vs. RS	0.73	**0.02**	0.76	**0.01**	0.73	**0.01**	0.66	**0.02**
SPEA2 vs. PAES	0.47	0.22	0.31	**0.01**	0.29	**0.01**	0.77	**0.01**
SPEA2 vs. SMPSO	0.49	0.31	0.36	**0.02**	0.36	**0.04**	0.59	0.07
SPEA2 vs. CellDE	0.32	**0.01**	0.28	**0.01**	0.35	**0.02**	0.68	**0.01**
SPEA2 vs. RS	0.73	**0.01**	0.55	0.13	0.47	0.22	0.81	**0.01**
PAES vs. SMPSO	0.47	0.17	0.52	0.23	0.54	0.13	0.52	0.25
PAES vs. CellDE	0.44	0.12	0.49	0.31	0.54	0.22	0.47	0.18
PAES vs. RS	0.75	**0.01**	0.74	**0.01**	0.66	**0.02**	0.61	**0.05**
SMPSO vs. CellDE	0.28	**0.01**	0.51	0.36	0.48	0.27	0.53	0.24
SMPSO vs. RS	0.69	**0.02**	0.73	**0.01**	0.77	**0.01**	0.70	**0.01**
CellDE vs. RS	0.76	**0.01**	0.74	**0.01**	0.70	**0.01**	0.69	**0.02**
(3) Product C60								
RWGA vs. NSGA-II	0.52	0.23	0.46	0.17	0.33	**0.02**	0.55	0.24
RWGA vs. MOCell	0.53	0.19	0.48	0.39	0.28	**0.01**	0.54	0.18
RWGA vs. SPEA2	0.48	0.30	0.47	0.14	0.27	**0.01**	0.41	0.06
RWGA vs. PAES	0.51	0.31	0.56	0.13	0.57	0.11	0.46	0.16
RWGA vs. SMPSO	0.46	0.17	0.49	0.32	0.52	0.27	0.46	0.13
RWGA vs. CellDE	0.34	**0.02**	0.49	0.37	0.56	0.13	0.44	0.22
RWGA vs. RS	0.59	**0.03**	0.62	**0.02**	0.74	**0.01**	0.66	**0.03**
NSGA-II vs. MoCell	0.54	0.23	0.49	0.32	0.63	**0.04**	0.54	0.22
NSGA-II vs. SPEA2	0.53	0.14	0.75	**0.01**	0.73	**0.01**	0.39	**0.03**
NSGA-II vs. PAES	0.57	0.09	0.51	0.33	0.74	**0.01**	0.57	0.11
NSGA-II vs. SMPSO	0.46	0.15	0.57	0.11	0.69	**0.02**	0.52	0.09
NSGA-II vs. CellDE	0.32	**0.05**	0.53	0.16	0.75	**0.01**	0.48	0.24

(*continued on next page*)

Table 13 (*continued*)

Pair of algorithms	Objectives							
	TMP: 0.8		*FPC*: 0.8		*FDC*: 0.85		*OET*: 2 h	
	A	*p*	*A*	*p*	*A*	*p*	*A*	*p*
NSGA-II vs. RS	0.57	**0.03**	0.65	**0.01**	0.81	**0.02**	0.59	**0.04**
MOCell vs. SPEA2	0.54	0.22	0.67	**0.02**	0.72	**0.01**	0.27	**0.01**
MOCell vs. PAES	0.48	0.24	0.72	**0.01**	0.51	0.30	0.56	0.09
MOCell vs. SMPSO	0.51	0.29	0.73	**0.01**	0.53	0.21	0.53	0.18
MOCell vs. CellDE	0.40	**0.05**	0.69	**0.02**	0.54	0.23	0.49	0.35
MOCell vs. RS	0.74	**0.01**	0.77	**0.01**	0.70	**0.01**	0.74	**0.01**
SPEA2 vs. PAES	0.55	0.17	0.36	**0.02**	0.35	**0.02**	0.65	**0.03**
SPEA2 vs. SMPSO	0.49	0.35	0.26	**0.01**	0.29	**0.01**	0.73	**0.01**
SPEA2 vs. CellDE	0.34	**0.02**	0.28	**0.01**	0.24	**0.01**	0.67	**0.01**
SPEA2 vs. RS	0.63	**0.05**	0.54	0.12	0.52	0.29	0.78	**0.02**
PAES vs. SMPSO	0.48	0.27	0.53	0.19	0.44	0.16	0.53	0.20
PAES vs. CellDE	0.20	**0.01**	0.47	0.31	0.53	0.14	0.48	0.33
PAES vs. RS	0.73	**0.01**	0.76	**0.01**	0.70	**0.01**	0.75	**0.01**
SMPSO vs. CellDE	0.27	**0.01**	0.53	0.29	0.48	0.20	0.55	0.17
SMPSO vs. RS	0.70	**0.01**	0.68	**0.02**	0.75	**0.01**	0.74	**0.01**
CellDE vs. RS	0.73	**0.01**	0.75	**0.01**	0.79	**0.01**	0.80	**0.01**
(4) Product C90								
RWGA vs. NSGA-II	0.22	**0.01**	0.36	**0.01**	0.44	0.16	0.33	**0.02**
RWGA vs. MOCell	0.53	0.16	0.45	0.14	0.26	**0.01**	0.53	0.17
RWGA vs. SPEA2	0.55	0.14	0.51	0.30	0.37	**0.04**	0.30	**0.02**
RWGA vs. PAES	0.52	0.19	0.49	0.34	0.56	0.14	0.59	0.08
RWGA vs. SMPSO	0.44	0.11	0.52	0.22	0.49	0.34	0.54	0.20
RWGA vs. CellDE	0.38	**0.02**	0.53	0.19	0.51	0.30	0.52	0.23
RWGA vs. RS	0.69	**0.01**	0.71	**0.01**	0.77	**0.01**	0.63	**0.02**
NSGA-II vs. MoCell	0.53	0.15	0.44	0.20	0.63	0.06	0.46	0.20
NSGA-II vs. SPEA2	0.49	0.37	0.68	**0.02**	0.77	**0.01**	0.31	**0.01**
NSGA-II vs. PAES	0.57	0.14	0.56	0.16	0.69	**0.02**	0.48	0.31
NSGA-II vs. SMPSO	0.49	0.33	0.57	0.09	0.73	**0.01**	0.55	0.14
NSGA-II vs. CellDE	0.38	0.06	0.54	0.17	0.70	**0.02**	0.51	0.33
NSGA-II vs. RS	0.91	**0.02**	0.93	**0.01**	0.81	**0.01**	0.81	**0.01**
MOCell vs. SPEA2	0.57	0.12	0.68	**0.02**	0.72	**0.01**	0.36	**0.02**
MOCell vs. PAES	0.52	0.21	0.77	**0.01**	0.54	0.14	0.49	0.31
MOCell vs. SMPSO	0.55	0.09	0.72	**0.01**	0.48	0.22	0.54	0.12
MOCell vs. CellDE	0.31	**0.02**	0.74	**0.01**	0.55	0.18	0.49	0.29
MOCell vs. RS	0.68	**0.02**	0.80	**0.01**	0.61	**0.05**	0.74	**0.01**
SPEA2 vs. PAES	0.54	0.16	0.28	**0.01**	0.34	**0.02**	0.75	**0.01**
SPEA2 vs. SMPSO	0.52	0.29	0.40	0.06	0.26	**0.01**	0.52	0.28
SPEA2 vs. CellDE	0.26	**0.01**	0.32	**0.01**	0.30	**0.01**	0.73	**0.01**
SPEA2 vs. RS	0.75	**0.01**	0.51	0.39	0.43	0.16	0.58	0.12
PAES vs. SMPSO	0.20	**0.01**	0.55	0.17	0.57	0.11	0.48	0.20
PAES vs. CellDE	0.54	0.19	0.44	0.10	0.59	0.08	0.54	0.13
PAES vs. RS	0.68	**0.02**	0.77	**0.01**	0.73	**0.01**	0.74	**0.01**
SMPSO vs. CellDE	0.32	**0.03**	0.55	0.24	0.53	0.28	0.49	0.29
SMPSO vs. RS	0.73	**0.01**	0.70	**0.01**	0.68	**0.02**	0.74	**0.01**
CellDE vs. RS	0.78	**0.01**	0.81	**0.01**	0.83	**0.01**	0.75	**0.01**

* *A*: $\hat{A}_{12}$, *p*: *p*-value. All *p*-values less than 0.05 are identified as bold.

corresponding to minimizing the test suites for various products. Table 16 shows the results of correlation between MFV_F_i and the increasing number of features, and between MFV_TC_j with the increasing associated test cases for various algorithms using Spearman's rank correlation (ρ). Recall that a higher value of MFV_F_i or MFV_TC_j represents a better performance of an algorithm.

Increasing Number of Features: For weight-based GAs (i.e., WBGA_W$_1$, WBGA_W$_2$, WBGA-MO and RWGA), we observed that the values of *MFV_F* increased significantly as the increasing number of features since the values for Spearman's ρ are greater than 0 (0.52, 0.47, 0.69 and 0.74 respectively) and the *p*-value is less than 0.0001, i.e., the performance increases significantly with the growth of features number. For SPEA2, PAES, SMPSO and CellDE, the values of *MFV_F* also increase but not significantly as the increasing features since Spearman's ρ is greater than 0 and *p*-values are greater than 0.05, i.e., the performances of SPEA2, PAES, SMPSO and CellDE increase but not significantly when the number of features increases. For NSGA-II, MOCell and RS, the performances decrease but not significantly as the increasing number of features since the Spearman's ρ is less than 0 and *p*-values are greater than 0.05. In summary, we conclude that RWGA has the ability to solve a wide range of problems with gradually better performance and the other seven algorithms are not influenced significantly by the number of features (Table 16).

Increasing Number of Associated Test Cases: Based on the results, we observed that for WBGA_W$_1$, WBGA_W$_2$, WBGA-MO, RWGA, SPEA2, PAES and SMPSO, the values of *MFV_TC* increase but not significantly as the number of associated test cases increases since Spearman's ρ is greater than 0 and *p*-values are greater than 0.05, i.e., the performances of WBGA_W$_1$, WBGA_W$_2$, WBGA-MO, RWGA, SPEA2, PAES and SMPSO increase but not significantly with the growth of associated test cases. For the other algorithms (NSGA-II, MOCell, CellDE and RS), the values of *MFV_TC* decrease but not significantly when the associated test cases increase, showing the performances of these algorithms decrease but not significantly as the increasing number of test cases. In summary, we can conclude even with the increase in the number of test cases, the performance of search algorithms is not influenced as shown in Table 16.

Table 14
Results for comparing the eight algorithms for OFV.*

Pair of algorithms	C20		C40		C60		C90	
	A	*p*	*A*	*p*	*A*	*p*	*A*	*p*
RWGA vs. NSGA-II	0.51	0.22	0.62	**0.01**	0.59	**0.04**	0.52	0.11
RWGA vs. MOCell	0.57	**0.03**	0.57	**0.02**	0.72	**0.01**	0.54	0.12
RWGA vs. SPEA2	0.55	0.07	0.62	**0.01**	0.69	**0.03**	0.64	**0.04**
RWGA vs. PAES	0.61	**0.02**	0.57	0.11	0.71	**0.01**	0.56	0.08
RWGA vs. SMPSO	0.71	**0.01**	0.67	**0.03**	0.64	**0.02**	0.55	0.09
RWGA vs. CellDE	0.56	0.14	0.75	**0.01**	0.60	0.06	0.68	**0.04**
RWGA vs. RS	0.66	**0.01**	0.79	**0.02**	0.65	**0.02**	0.72	**0.02**
NSGA-II vs. MoCell	0.59	0.11	0.51	0.23	0.61	0.09	0.53	0.19
NSGA-II vs. SPEA2	0.61	**0.04**	0.69	**0.04**	0.65	0.09	0.78	**0.02**
NSGA-II vs. PAES	0.58	0.15	0.56	0.17	0.61	0.07	0.45	0.22
NSGA-II vs. SMPSO	0.45	0.33	0.61	**0.04**	0.52	0.32	0.55	0.18
NSGA-II vs. CellDE	0.56	0.25	0.57	0.08	0.51	0.29	0.63	**0.04**
NSGA-II vs. RS	0.64	**0.02**	0.69	**0.01**	0.58	**0.01**	0.70	**0.02**
MOCell vs. SPEA2	0.65	**0.02**	0.73	**0.04**	0.55	0.09	0.64	**0.04**
MOCell vs. PAES	0.58	0.08	0.55	0.12	0.51	0.21	0.57	0.09
MOCell vs. SMPSO	0.59	0.14	0.56	0.13	0.52	0.08	0.51	0.14
MOCell vs. CellDE	0.49	0.21	0.59	0.17	0.60	0.19	0.53	0.19
MOCell vs. RS	0.71	**0.02**	0.75	**0.01**	0.67	**0.04**	0.66	**0.02**
SPEA2 vs. PAES	0.34	**0.02**	0.31	**0.02**	0.43	0.09	0.29	**0.01**
SPEA2 vs. SMPSO	0.28	**0.01**	0.32	**0.02**	0.38	**0.02**	0.41	0.07
SPEA2 vs. CellDE	0.45	0.11	0.32	**0.03**	0.28	**0.02**	0.27	**0.02**
SPEA2 vs. RS	0.55	0.32	0.51	0.24	0.48	0.12	0.54	0.08
PAES vs. SMPSO	0.48	0.17	0.51	0.09	0.43	0.17	0.52	0.25
PAES vs. CellDE	0.46	0.21	0.45	0.14	0.52	0.15	0.41	0.09
PAES vs. RS	0.65	**0.02**	0.72	**0.02**	0.66	**0.03**	0.71	**0.01**
SMPSO vs. CellDE	0.49	0.32	0.46	0.29	0.43	0.16	0.38	0.06
SMPSO vs. RS	0.63	**0.01**	0.77	**0.01**	0.64	**0.03**	0.69	**0.02**
CellDE vs. RS	0.72	**0.01**	0.75	**0.01**	0.68	**0.02**	0.75	**0.01**

* A: $\hat{A}_{12}$, p: p-value. All p-values less than 0.05 are identified as bold.

6.2.3. Timing analysis for RWGA and the other selected search algorithms

Similarly, we also performed a timing analysis for the best weight-based GAs (i.e., RWGA) and the other kinds of selected search algorithms (e.g., NSGA-II) as compared with RS. Table 17 shows the average running time per run taken by each selected search algorithms. We also obtained a p-value of 0.63 with the Kruskal–Wallis test suggesting there are no significant differences between the running time of the selected search algorithms as compared with RS. Based on the results, we can conclude that all the selected search algorithms did not take significantly different time for running as compared with RS in terms of finding an optimal solution.

Table 15
Results for the Vargha and Delaney statistics-artificial problems (without/with Whitney *U* test).

RQ	Pair of Algorithms (A vs. B)	A > B	A < B	A = B
RQ2	RWGA vs. RS	489/480	11/0	0/20
	NSGA-II vs. RS	467/449	33/14	0/37
	MOCell vs. RS	438/419	62/46	0/35
	SPEA2 vs. RS	298/251	202/179	0/70
	PAES vs. RS	430/414	70/59	0/27
	SMPSO vs. RS	388/362	112/93	0/45
	CellDE vs. RS	423/399	77/65	0/36
RQ3	RWGA vs. NSGA-II	432/405	68/57	0/32
	RWGA vs. MOCell	403/397	97/85	0/18
	RWGA vs. SPEA2	464/451	36/23	0/26
	RWGA vs. PAES	381/362	119/104	0/24
	RWGA vs. SMPSO	453/438	47/39	0/23
	RWGA vs. CellDE	380/357	120/103	0/40
	NSGA-II vs. MOCell	269/252	231/204	0/44
	NSGA-II vs. SPEA2	398/377	102/86	0/47
	NSGA-II vs. PAES	266/242	234/219	0/39
	NSGA-II vs. SMPSO	226/219	273/241	0/40
	NSGA-II vs. CellDE	245/227	255/239	0/34
	MOCell vs. SPEA2	395/374	105/91	0/35
	MOCell vs. PAES	302/281	198/167	0/52
	MOCell vs. SMPSO	267/243	233/200	0/57
	MOCell vs. CellDE	201/182	299/283	0/35
	SPEA2 vs. PAES	170/145	330/307	0/48
	SPEA2 vs. SMPSO	141/107	359/316	0/77
	SPEA2 vs. CellDE	179/152	321/303	0/45
	PAES vs. SMPSO	223/201	277/255	0/46
	PAES vs. CellDE	198/182	302/270	0/48
	SMPSO vs. CellDE	244/219	256/238	0/43

7. Discussion

In this section, we discuss the results of the experiments based on our industrial case study and the 500 artificial problems. Section 7.1 provides insight on the results of RQ1–RQ3, while Section 7.2 discusses results of RQ4. Recall that RQ1–RQ3 address questions related to cost/effectiveness and performance of our minimization problem (as compared with RS), while RQ4 refers to the impact of the increasing number of features and the number of test cases on the performance of the selected multi-objective search algorithms (Section 5.1.1). Before getting deeper into the specific discussion related with the research questions, the key results are first summarized as below:

- Different algorithms achieve the best performance for each objective, i.e., CellDE for *TMP*, MOCell for *FPC*, NSGA-II for *FDC* and SPEA2 for *OET*;
- RWGA achieves significantly best performance among all the selected search algorithms when considering all the objectives together;
- The performance of weight-based GAs (e.g., RWGA) increases with the growth of features, but is not significantly influenced by the increasing number of associated test cases;
- All the other search algorithms are not significantly impacted by the increasing complexity of problems (increasing features and associated test cases).

Table 16
Spearman's correlation analysis for the select algorithms with the increasing number of features and associated test cases.

Algorithms	Increasing number of features		Increasing number of associated test cases	
	Spearman ρ	Prob > $\|\rho\|$	Spearman ρ	Prob > $\|\rho\|$
WBGA_W$_1$	0.52	**<0.0001**	0.08	0.5764
WBGA_W$_2$	0.47	**<0.0001**	0.10	0.5123
WBGA-MO	0.69	**<0.0001**	0.15	0.4892
RWGA	0.74	**<0.0001**	0.19	0.4719
NSGA-II	−0.13	0.5546	−0.10	0.6714
MOCell	−0.11	0.6852	−0.02	0.7653
SPEA2	0.09	0.8019	0.12	0.5417
PAES	0.02	0.7980	0.14	0.4928
SMPSO	0.15	0.5742	0.07	0.6986
CellDE	0.08	0.7006	−0.13	0.3985
RS	−0.08	0.8704	−0.09	0.7619

7.1. Discussion related to RQ1–RQ3

Weight-based GAs require sets of weights (e.g., w_1, w_2 and w_3) that can be provided by domain experts based on the testing requirements. Our results and analysis show that RWGA presents the best performance among the three weight-based GAs and RS. In fact, RWGA assigns weights dynamically during the search and thus the search is more efficiently guided toward the best weights, to reach the threshold values for TMP, FPC, FDC and OET.

When comparing the best weight-based GA (RWGA) and the other search algorithms, from our industrial case study, we observe that the solutions obtained by RWGA, NSGA-II, MOCell, PAES, SMPSO, CellDE can reach the expected thresholds for each objective (Section 6.2.1). SPEA2 managed to reach the expected thresholds for *TMP* and *OET* but not for *FPC* and *FDC*. RS cannot find optimal solutions for the thresholds (RQ1). When comparing the selected search algorithms with RS, all the algorithms except SPEA2 have significantly better performance than RS in finding optimal solutions for each objective function and *OFV* considering all the objectives. SPEA2 has significantly better performance than RS for *TMP* and *OET* but has almost equivalent performance as RS for *FPC*, *FDC* and *OFV* (RQ2).

Moreover, when studying the performance of each individual algorithm, different algorithms achieve the best performance for different objectives (e.g., CellDE for *TMP*, MOCell for *FPC*, NSGA-II for *FDC* and SPEA2 for *OET*). However, when combining all objectives together, RWGA has significantly better performance than all the other search algorithms (RQ3). For the artificial problems (RQ1–RQ3), we observe consistent results with our observations on the industrial case study. This difference in performance can be explained as follows: RWGA aims at balancing all the required objectives and finding an optimal solution by considering all the objectives together. Moreover, by dynamically changing weights at each generation, RWGA more thoroughly explores the search space of weights. This dynamic assignment of the weights (i.e., RWGA) is promising to find an optimal solution in a given number of generations, as compared to a strategy using only fixed weights for all the generations (e.g., WBGA_W$_1$ and WBGA_W$_2$). Recall from Section 2 that NSGA-II, MOCell, SPEA2, PAES, SMPSO and CellDE are used to seek a set of optimal non-dominated solutions. However, when considering all the objectives together (*OFV*), such non-dominated solutions may be worse than the solutions found by RWGA that aims at balancing all the objectives together. Notice that the solution obtained by RWGA may not dominate the solutions obtained by the other search algorithms (e.g., NSGA-II) but RWGA can manage to find the best solution with maximum overall fitness function (*OFV*) for the minimization problem. However, if users only concern one of the most important objectives (e.g., *FDC*), our approach with corresponding tool (Section 8) will recommend the most suitable algorithm accordingly (e.g., MOCell for *FDC*).

Table 17
Average running time for the selected weight-based search algorithms.

Algorithms	RWGA	NSGA-II	MOCell	SPEA2	PAES	SMPSO	CellDE	RS
Time (s)	1.37	1.54	1.23	1.98	2.06	2.19	1.45	1.13

7.2. Discussion related to RQ4

When looking into the performance of each individual algorithm (RQ4) as the number of features and test cases increase (MFV_F and MFV_TC as shown in Figs. 2 and 3), we observe that RWGA presents the best performance and has the ability to solve a wider range of problems with gradually better performance (Fig. 2). In

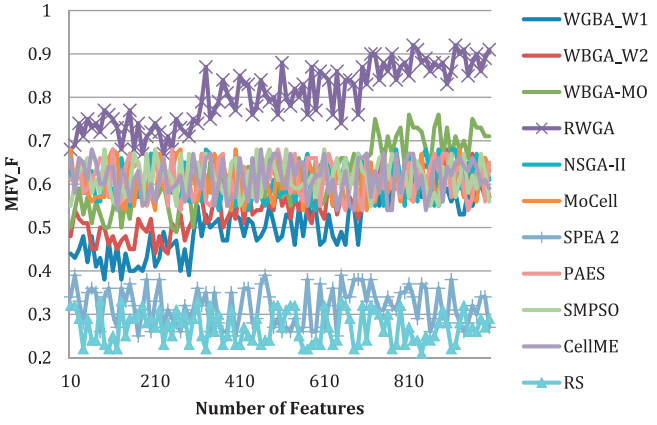


Fig. 2. Mean fitness values for number of features.

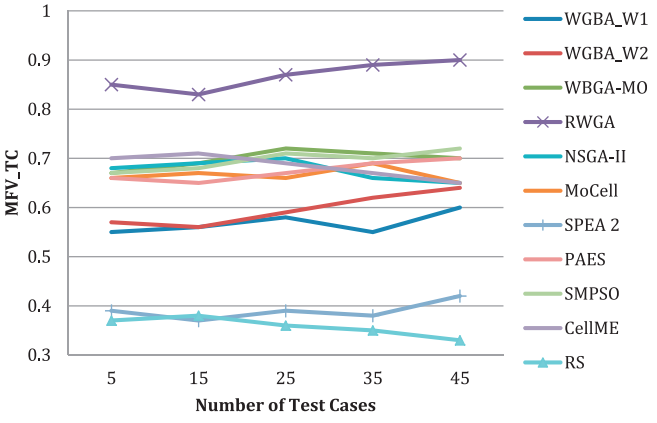


Fig. 3. Mean fitness values for number of test cases.

addition, when the number of test cases associated to each feature increases, the performance of RWGA is not significantly impacted (Fig. 3).

To show the combined effect of the increase in the number of features and test cases on the performance of search algorithms, we show surface plots based on the mean fitness value (MFV, Section 5.1.3) in Fig. 4. From Fig. 4, we observe that RWGA reaches the best performance among all the algorithms. Moreover, the performance of WBGA and WBGA-MO also increases as the number of features increases. Finally, the performance of the other seven algorithms is not significantly impacted (notice that this observation is consistent with the results presented in Figs. 2 and 3). For instance, the performance of NSGA-II stays stable as the increasing number of the features and associated test cases. It is interesting to observe that the performance of SPEA2 remains worse than the other search algorithms without significantly changing with the growth of features and associated test cases. Notice that RS performs worse among all the selected search algorithms. These observations suggest that the selected search algorithms together with our proposed fitness function are scalable and sufficient for test minimization for the products of varying complexity. Sections 7.2.1 and 7.2.2 further provide detailed explanations about the performance when the number of features and test cases increases.

7.2.1. Increasing number of features

Three different types of solution spaces need to be first introduced to understand the rest of the section:

(1) The *Solution Space* (black ellipse in Fig. 5) refers to the space of potential solutions for any search algorithm, depending on the number of test cases. When the number of features increases, then the solution space can either: (A) increase when more test cases associated with new features are introduced; or (B) remain the same when no more test cases are introduced to the solution space (i.e., the existing test cases can cover the new features). For instance, in our industrial case study (Cisco), two call protocols *H323* and *SIP* are designed and relevant test cases are developed for testing *H323* and *SIP*. If a new call protocol based on *H323* and *SIP* is designed, the existing test cases may be reused for testing such new protocol. In this case, no new test cases will be introduced to the solution space.
(2) The *Balanced Solution Space* (red ellipse in Fig. 5) represents the space for optimal solutions, balancing all the objectives together. Weight-based GAs aims at guiding the search within this subspace in order to find optimal balanced solutions. When the number of features increases, the balanced solution space can either (A) grow, (B) remain the same or (C) decrease.
(3) The *Non-dominated Solution Space* (green ellipse in Fig. 5) refers to the space for non-dominated optimal solutions. The algorithms based on dominance theory (i.e., NSGA-II, MOCell, SPEA2, PAES, SMPSO and CellDE) can guide the search to find a set of non-dominated solutions within this solution space. Similarly, the non-dominated solution space can either (A) grow, (B) remain the same, or (C) decrease with the growth of features.

Based on the above-mentioned definitions, our observations for experiments can be explained under the situation when the solution space, the balanced solution space and the non-dominated solution all grow as the increasing number of features. More specifically, the gradual improvement of performance of weight-based GAs (WBGA, WBGA-MO and RWGA) is interpreted as follows: Weight-based GAs (e.g., RWGA) guides the search to find optimal balanced solutions, considering all the objectives. Hence, the space of balanced solutions grows when the space for potential solutions grows. As a result, weight-based GAs (i.e., RWGA) easily finds an

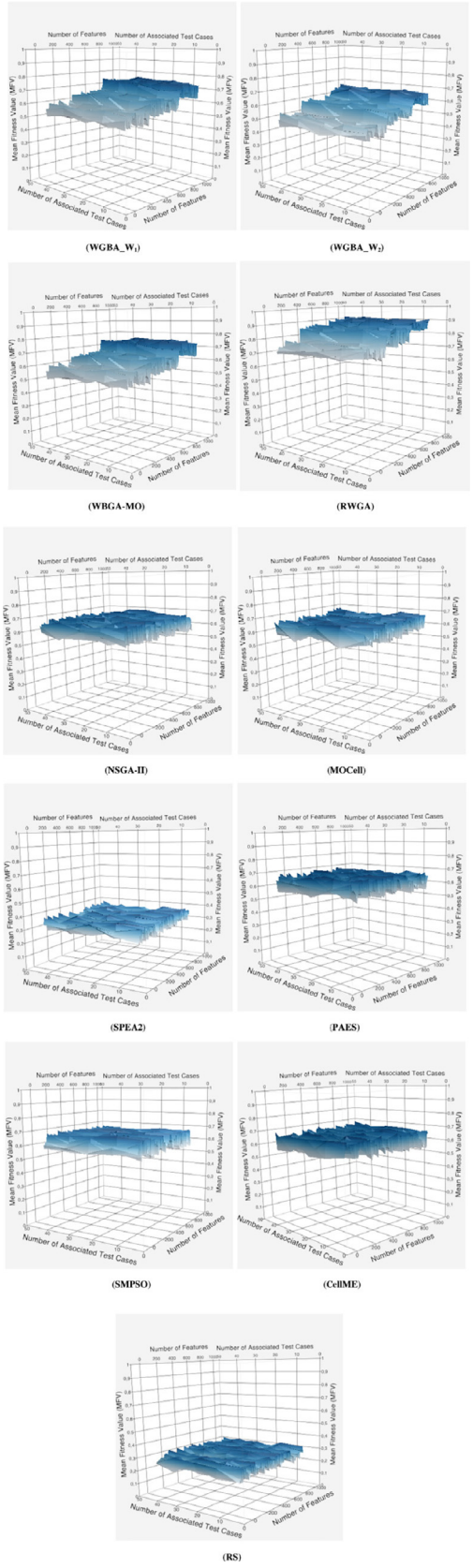


Fig. 4. Surface plots for all the selected algorithms.

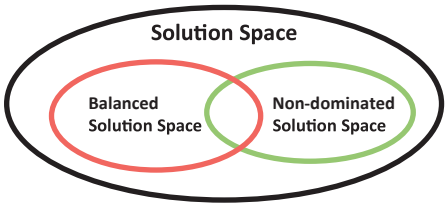


Fig. 5. An overall diagram for three types of space.

optimal balanced solution from a larger balanced solution space with the aim for balancing all the objectives together.

For the other algorithms based on the dominance theory (i.e., NSGA-II, MOCell, SPEA2, PAES, SMPSO and CellDE), the performance is not significantly influenced by increasing number of features since these algorithms guide the search in finding a set of non-dominated solutions (usually one objective is the main concern). The overall fitness value (*OFV*, Section 5.1.3) does not change significantly even when the space for non-dominated solutions increases, as shown in (a) and (b) of Fig. 5. For instance, solution *a* and solution *b* are two obtained non-dominated solutions by NSGA-II with $TMP = 0.8$, $FPC = 0.5$, $FDC = 0.5$, $OET = 0.5$, $AEF = 0.5$ and $TMP = 0.5$, $FPC = 0.8$, $FDC = 0.5$, $OET = 0.5$, $AEF = 0.5$, respectively, i.e., the main objective for solution *a* is the percentage of test minimization (*TMP*) while the main concern for solution *b* is feature pairwise coverage (*FPC*). But for the overall fitness value (*OFV*) when considering all the objectives, solution *a* and *b* are equivalent since the *OFV* values for both solutions are the same (i.e., both values of *OFV* for solution *a* and solution *b* are 0.56, Section 5.1.3), which is the main reason the performance of NSGA-II, MOCell, SPEA2, PAES, SMPSO and CellDE stays stable as shown in Figs. 2 and 4.

For random search (RS), the performance remains the worst and is not influenced by the increasing features since it chooses solutions randomly without any guideline.

For the other combinations for the solution space, the balanced solution space and the non-dominated solution, the observations for experiments may not be the same as what we observed in our experiments. This further requires more focused experiments that assess the performance of the search algorithms with varying combinations of increase, decrease, and no change in the three types of solution spaces.

7.2.2. Increasing number of associated test cases

When increasing the number of test cases, we observed no significant impact on the performance for all the selected algorithms, which can be explained as follows: The complexity of problems mainly depends on the number of features (*FPC* measures how many pairs are covered, but the number of test cases is ignored) and the relationships between features and test cases are not complicated in our case (i.e., the only relationship between features and associated test cases is that one feature can be associated with a certain number of test cases as discussed in Section 4). As a result, when the complexity of a problem is mainly determined by the number of features, the number of associated test cases may have no significant impact on the performance of algorithms as shown in Figs. 3 and 4. However, in other cases, more complicated relationships may exist between features and associated test cases, e.g., testing some features requires several steps and each step is covered by a certain number of test cases or execution of test cases depends on the execution of other test cases. Thus, the number of associated test cases may have significant impact on the performance of algorithms. We plan to further investigate this problem by using different case studies and other focused experiments in the future.

Concluding remarks: When multi-objective test minimization for software product lines is sought, we recommend using CellDE, MOCell, NSGA-II and SPEA2 for the objectives *TMP*, *FPC*, *FDC* and *OET*, respectively, when one objective has more importance than the others. In contrast, using RWGA is recommended when optimizing all the objectives together is sought. The latter situation happened in our industrial case study.

8. Automation

In this section, we present the tool support for test minimization using search algorithms along with our proposed fitness functions. The tool is called TEst Minimization with Search Algorithms (TEMSA) developed to minimize the test cases at the same time improving effectiveness and reducing predetermined cost based on various cost/effectiveness measures. TEMSA is implemented as a web-based application on top of the jMetal library using Java Sencha ExtJS (a JavaScript Framework for Rich Web Apps) (Sencha, 2010). The architecture of TEMSA is shown in Fig. 6 (TEMSA can be tried out at the website: http://zen-tools.com/TEMSA/).

The input of TEMSA is a set of selected features and test cases for testing a product as an XML file in our industrial case study. To use TEMSA, users have to provide an input file corresponding to our specific XML schema (the related XML schema and sample input XML file can be downloaded from our tool website). Cost measures (e.g., *OET*) and effectiveness measures (e.g., *TMP*, *FPC*, *FDC* and *AEF*) are integrated into TEMSA for covering various objectives. Other objective measures can also be coded into TEMSA by formulating them as specific objective functions mathematically as we discussed in Section 3.2. For the test cases, each one includes three key attributes for cost/effectiveness measures, i.e., *SucR*, *EF* and *AET* (Section 3.2), which are obtained from test execution history. Based on these inputs, TEMSA outputs a set of minimized test cases organized as another XML file which also conforms to our specific XML schema and can be transformed into any other schema when needed. In our context, the output can be put into test execution engine for scheduling and further executing for System Under Test (SUT). Afterwards, the output of executed test cases will be an input to update test execution history in the repository, such as *SucR*, *EF* and *AET* for the executed test cases.

Moreover, our tool recommends different algorithms depending on test minimization objectives based on the results of our empirical study as shown in Fig. 7. First, a user selects an objective with highest importance (*Activity A1*) and TEMSA will automatically recommend the relevant best algorithm based on the selected objective. For instance, if the highest importance is to reduce the number of test cases (*TMP*), then CellDE will be recommended by TEMSA (*Activity A2*). Similarly, if feature pairwise coverage (*FPC*), fault detection capability (*FDC*) and the allowed time budget (*OET*) are the most important objectives, TEMSA will recommend users to adapt MOCell, NSGA-II and SPEA2 for their test minimization, respectively (*Activity A3*, *A4* and *A5*). Moreover, if a user wants to balance all the objectives together for test minimization, they do not need to select any objective, (*Activity A1*) and RWGA will be recommended by TEMSA automatically (*Activity A6*).

9. Threats to validity

In this section, we discuss four threats to validity for our experiments and how we address them.

9.1. Internal validity

Internal validity threats exist when the outcome of results is influenced by internal factors (e.g., parameters of search

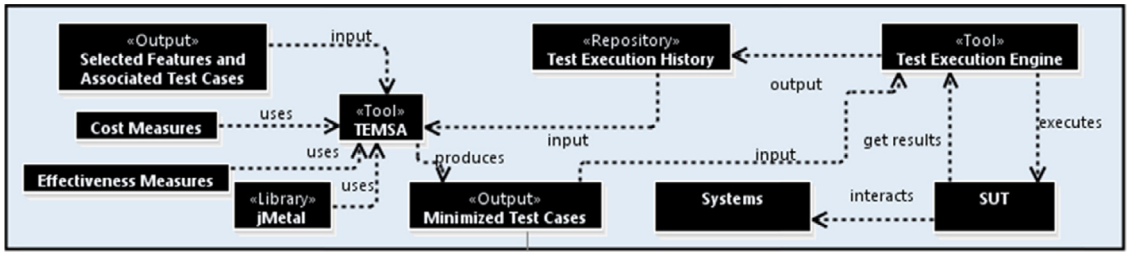


Fig. 6. Architecture for TEMSA

algorithms) and are not necessarily due to the application of the treatment (e.g., search algorithms) being studied (Ali et al., 2010; Barros and Neto, 2011). A possible threat to *internal validity* is that we have experimented with only one-default configuration settings for the parameters of the selected search algorithms. However, these settings are in accordance with the common guidelines in the literature and our previous experience on testing problems (Arcuri and Briand, 2011; Sheskin, 2007; Wang et al., 2013c).

9.2. Conclusion validity

Conclusion validity threats are concerned with factors that can influence the conclusion that can be drawn from the results of the experiments. The most probable *conclusion validity* threat in experiments involving randomized algorithms is due to random variations. To address it, we repeated experiments 100 times to reduce the possibility the results were obtained by chance. Furthermore, to determine the probability of yielding higher performance by different algorithms, we measured the effect size using Vargha and Delaney statistics test since it is appropriate for non-parametric effect size measure, which matches our situation. Meanwhile, we performed Mann–Whitney *U* test to determine the statistical significance of the results (Arcuri and Briand, 2011). Finally, Spearman's rank correlation coefficient is used to measure the potential impact on the performance of selected search algorithms by increasing number of features and associated test cases since it is mainly used for non-parametric correlation coefficient measure and suits our objective in this paper (Yue et al., 2013).

9.3. Construct validity

For the *construct validity* threats, we looked at the validity of the comparison measures for all the selected search algorithms. The most frequently observed threat was using some measures of cost that have severe limitations, as they are not precise. To reduce *construct validity* threats in our context, we used the same stopping criteria for all algorithms, i.e., number of fitness evaluations. We ran each algorithm for 25,000 evaluations to seek the best solution for test minimization. This criterion is a comparable measure across all the algorithms since each iteration requires updating the obtained solution and comparing the computed value of fitness function.

9.4. External validity

External validity threats exist when the outcome of results is influenced by external factors and are not necessarily due to the application of the treatment being studied (Ali et al., 2010; Barros and Neto, 2011). One common *external validity* threat in the software engineering experiments is about generalization of results. One may argue the results cannot be generalized for other case studies if we ran our experiments on just one industrial case study. However, such threat to external validity is common to all empirical studies. In addition, we also conducted an empirical evaluation using carefully designed 500 artificial problems and the results are consistent with the industrial case study.

10. Related works

In the context of regression testing, the optimization problem can be classified into the following three problems, i.e., test case selection, minimization, and prioritization (Fischer et al., 1981; Harman, 2011; Yoo and Harman, 2007, 2012; Zheng et al., 2007). More specifically, test case selection focuses on selecting a subset of test cases from the existing test suite with aim to test a modified version of systems or programs while test minimization eliminates the redundant test cases from the existing test suite for the current systems or programs in order to reduce the cost of testing

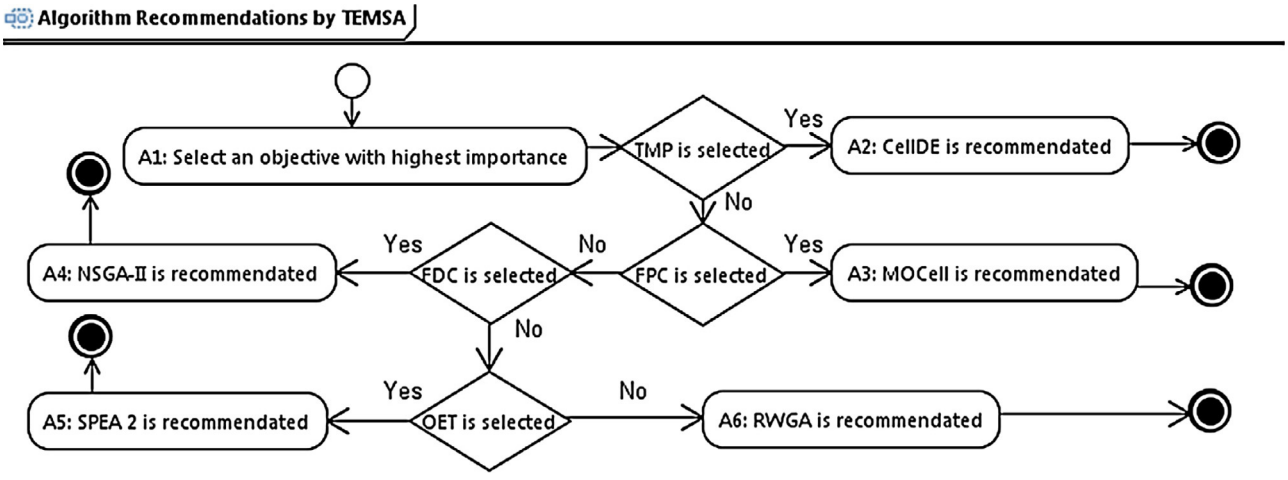


Fig. 7. Recommendations for algorithms by TEMSA

(e.g., time). Test prioritization orders the test cases in the existing test suite to test the current systems or programs with the objective of achieving the pre-defined criteria (e.g., maximum number of executing test cases) in a given time budget. The main difference between selection and minimization is that test minimization eliminates the redundant test cases permanently for the systems (programs), while test case selection refers to select relevant test cases temporarily for testing the modified version of the systems (Harman, 2011; Yoo and Harman, 2012). However, from the perspective of optimization, there is no significant difference between test case selection and test minimization (Harman, 2011), i.e., both of them aim at choosing a subset of test cases from the existing test suite at the same time achieving pre-defined objectives for optimization, which was called "test case selection" problem in (Harman, 2011).

Our work is related to the above-mentioned selection problem, but as opposed to the existing works, our work lies in the context of product line testing (e.g., feature pairwise coverage is used to measure the effectiveness of the minimized test suite based on the coverage of feature pairs from the product line view rather than code level (e.g., code coverage) in the most of the existing works for regression testing). Although there exist a large number of test selection/minimization techniques in the context of regression testing, there is not enough evidence that these techniques can be adapted to product lines (e.g., whether code coverage in regression testing is equivalent to the feature coverage in product line testing).

Yoo and Harman (2007) used a greedy algorithm and a multi-objective search algorithm NSGA-II for test case selection for the following three measures: code coverage, fault detection history, and execution time in the context of regression testing. Notice that these three measures are similar to our cost/effectiveness measures *FPC*, *FDC* and *OET*. The difference with our work is that first we select a set of relevant test cases for a new product using feature models as we discussed in (Wang et al., 2012, 2013a) and then we perform test minimization based on two additional measures that are *TMP* and *AEF*. In addition, we conducted an empirical study to compare ten multi-objective algorithms belonging to four different mechanisms and evaluate their performance, which was not addressed in Yoo and Harman (2007).

In another work related to test prioritization in regression testing (Walcott et al., 2006), a two-objective problem (i.e., code coverage and execution time) is converted into a single-objective problem using an arithmetical combination of weights for the fitness function. These two objectives are close to our cost/effectiveness measures *FPC* and *OET*. However, as compared with code coverage, our defined *FPC* mainly focuses on the coverage of feature pairs from the view of product line rather than code level of the systems. Moreover, additional three measures are defined mathematically in our work and we do not limit our scope within weight-based GAs, i.e., different types of search algorithm are empirically evaluated, e.g., NSGA-II and SPEA2.

As compared with our previous work (Wang et al., 2013c), we define two additional cost/effectiveness measures (i.e., *AEF* and *OET*) for test minimization problem. Moreover, we further compare the performances for the best weight-based GA (i.e., RWGA) and the other six different search algorithms using an industrial case study and 500 designed artificial problems, which cover various kinds of search algorithms and make the results and analysis more persuasive.

To the best of our knowledge and existing published reviews (Harman et al., 2009; Yoo and Harman, 2012), none of the existing works studied test minimization in the context of product line considering *TMP*, *FPC*, *FDC*, *OET* and *AEF* all together. In addition, it is one of the few works, which extensively compares the performance of search algorithms belonging to different mechanisms (e.g., evolutionary algorithms, swarm algorithms, and hybrid algorithms) for test suite minimization in the context of product lines.

11. Conclusion and future work

In this paper, we have proposed and have evaluated a fitness function in conjunction with ten multi-objective search algorithms, to minimize test suites in the context of product line testing. Our fitness function takes effectiveness measures (i.e., Test Minimization Percentage (*TMP*), Feature Pairwise Coverage (*FPC*), Fault Detection Capability (*FDC*) and Average Execution Frequency (*AEF*)) and cost measures (i.e., Overall Execution Time (*OET*)) into account to guide the search. The search algorithms, along with our fitness function, are evaluated on an industrial case study and 500 artificial problems of varying size and complexity using a tool we have developed for this purpose, called TEst Minimization with Search Algorithms (TEMSA).

Given a set of thresholds for each objective, the results show that, unlike RS, WBGA-MO, RWGA, NSGA-II, MOCell, PAES, SMPSO and CellDE can reach these thresholds, while SPEA2 achieves acceptable performance for *TMP*/*OET*, which is not the case for *FPC*/*FDC*. Meanwhile, we observe that the performance of WBGA (i.e., WGBA_W_1 and WGBA_W_2) is not stable since it largely depends on the predetermined weights for the objectives. When comparing the selected algorithms, our results show that WBGA, WBGA-MO, RWGA, NSGA-II, MOCell, PAES, SMPSO and CellDE significantly outperform RS and SPEA2 has almost equivalent performance than RS. From the results on artificial problems, we conclude that RWGA also achieves the best performance when taking all objectives into account. Moreover, RWGA can solve a wider range of problems and its performance is improved when the number of features number is growing. Notice the performance of other weight-based GA is also significantly increased with the growth of features. For rest of the algorithms, the performance is not significantly influenced by the increasing number of features or test cases, when it comes to finding an optimal solution.

Our short-term future plan includes the analysis of another industrial case study in order to complement our experimental results and refine the design of TEMSA. We also plan to refine the fitness function by considering other objectives, such as the number of available computing resources, and evaluate other major search algorithms, such as Archive-Based hYbrid Scatter Search (AbYSS), which combines both scatter search and genetic algorithms.

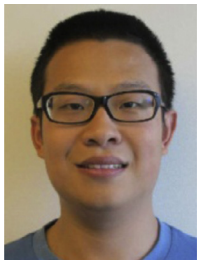
Acknowledgements

We gratefully thank Marius Liaaen and his team at Cisco for providing us the valuable industrial case study. All the authors acknowledge funding from the Research Council of Norway.

References

Ali, S., Briand, L.C., Hemmati, H., Panesar-Walawege, R.K., 2010. A systematic review of the application and empirical investigation of search-based test case generation. IEEE Trans. Softw. Eng. 36, 742–762.
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.
Barros, M.d.O., Neto, A.C.D., 2011. Threats to Validity in Search-based Software Engineering Empirical Studies. UNIRIO – Universidade Federal do Estado do Rio de Janeiro 0006/2011.
Benavides, D., Segura, S., Ruiz-Cortés, A., 2010. Automated analysis of feature models 20 years later: a literature review. Inf. Syst. 35, 615–636.
Brownlee, J., 2012. Clever Algorithms: Nature-Inspired Programming Recipes. lulu.com, 1st ed..
CiscoSystems, C., 2010. Cisco Telepresence Codec c90, Data Sheet..
Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. 6, 182–197.

Durillo, J., Nebro, A., Luna, F., Alba, E., 2008. Solving Three-objective optimization problems using a new hybrid cellular genetic algorithm. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (Eds.), Parallel Problem Solving from Nature – PPSN X. Springer, Berlin Heidelberg, pp. 661–670.
Durillo, J.J., Nebro, A.J., 2011. jMetal: a Java framework for multi-objective optimization. Adv. Eng. Softw. 42, 760–771.
Engström, E., Runeson, P., 2011. Software product line testing – a systematic mapping study. Inf. Softw. Technol. 53, 2–13.
Fischer, K.F., Raji, F., Chruscicki, A., 1981. A methodology for retesting modified software, National Telecommunications Conference (NTC), pp. 1–6.
GmbH, P., 2006. Variant Management with Pure::Variants. Technical White Paper.
Greer, D., Ruhe, G., 2004. Software release planning: an evolutionary and iterative approach. Inf. Softw. Technol. 46, 243–253.
Harman, M., 2011. Making the case for MORTO: multi objective regression test optimization. In: Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops. IEEE Computer Society, pp. 111–114.
Harman, M., Mansouri, S.A., Zhang, Y., 2009. Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications. Technical Report TR-09-03.
Knowles, J.D., Corne, D.W., 2000. Approximating the nondominated front using the pareto archived evolution strategy. Evol. Comput. 8, 149–172.
Konak, A., Coit, D.W., Smith, A.E., 2006. Multi-objective optimization using genetic algorithms: a tutorial. Reliab. Eng. Syst. Saf. 91, 992–1007.
Kuo-Chung, T., Yu, L., 2002. A test generation strategy for pairwise testing. IEEE Trans. Softw. Eng. 28, 109–111.
Lopez-Herrejon, R.E., Chicano, F., Ferrer, J., Egyed, A., Alba, E., 2013. Multi-objective optimal test suite computation for software product line pairwise testing, International Conference on Software Maintenance (ICSM), pp. 404–407.
Mcdonald, J.H., 2009. Handbook of Biological Statistics.
McGregor, J., 2001. Testing a Software Product Line (CMU/SEI-2001-TR-022). Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.
Nebro, A., Durillo, J., Luna, F., Dorronsoro, B., Alba, E., 2007. Design issues in a multiobjective cellular genetic algorithm. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (Eds.), Evolutionary Multi-Criterion Optimization. Springer, Berlin Heidelberg, pp. 126–140.
Nebro, A.J., Durillo, J.J., Garcia-Nieto, J., Coello Coello, C.A., Luna, F., Alba, E., 2009. SMPSO: a new PSO-based metaheuristic for multi-objective optimization, MCDM'09. IEEE Symposium on Computational intelligence in Multi-criteria Decision-Making, 2009, pp. 66–73.
Runeson, P., Engström, E., 2012. Regression testing in software product line engineering. Adv. Comput. 86, 223–263.
Sencha, 2010. http://www.sencha.com/products/extjs..
Sheskin, D.J., 2007. Handbook of Parametric and Nonparametric Statistical Procedures. Chapman&Hall&CRC.
Walcott, K.R., Soffa, M.L., Kapfhammer, G.M., Roos, R.S., 2006. TimeAware test suite prioritization, In: Proceedings of the 2006 International Symposium on Software Testing and Analysis. ACM, Portland, Maine, USA, pp. 1–12.
Wang, S., Ali, S., Gotlieb, A., 2013. Minimizing test suites in software product lines using weight-based genetic algorithms, In: Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference. ACM, Amsterdam, The Netherlands, pp. 1493–1500.
Wang, S., Gotlieb, A., Ali, S., Liaaen, M., 2013. Automated test case selection using feature model: an industrial case study. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (Eds.), Model-Driven Engineering Languages and Systems. Springer, Berlin Heidelberg, pp. 237–253.
Wang, S., Gotlieb, A., Liaaen, M., Briand, L.C., 2012. Automatic selection of test execution plans from a video conferencing system product line, In: Proceedings of the VARiability for You Workshop: Variability Modeling Made Useful for Everyone. ACM, Innsbruck, Austria, pp. 32–37.
Yoo, S., Harman, M., 2007. Pareto efficient multi-objective test case selection, In: Proceedings of the 2007 International Symposium on Software Testing and Analysis. ACM, London, United Kingdom, pp. 140–150.
Yoo, S., Harman, M., 2012. Regression testing minimization, selection and prioritization: a survey. Softw. Test. Verif. Reliab. 22, 67–120.
Yue, T., Briand, L.C., Labiche, Y., 2013. Facilitating the transition from use case models to analysis models: approach and experiments. ACM Trans. Softw. Eng. Methodol. 22, 1–38.
Zhang, Y., Finkelstein, A., Harman, M., 2008. Search based requirements optimisation: existing work and challenges. In: Paech, B., Rolland, C. (Eds.), Requirements Engineering: Foundation for Software Quality. Springer, Berlin Heidelberg, pp. 88–94.
Zheng, L., Harman, M., Hierons, R.M., 2007. Search algorithms for regression test case prioritization. IEEE Trans. Softw. Eng. 33, 225–237.
Zitzler, E., Laumanns, M., Thiele, L., 2001. SPEA2: improving the strength pareto evolutionary algorithm, The EUROGEN 2001-Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, pp. 95–100.



Shuai Wang obtained his master degree for computer science and technology in the field of software testing from Beihang University, Beijing, China. He is currently working as PhD student at the Certus Software Verification & Validation Center hosted by Simula Research Laboratory and the Department of Informatics, University of Oslo, Norway. His main research interests are: variability modeling, model-based testing and search-based testing. He is a student member of the ACM and IEEE computer society.

Shaukat Ali is currently a research scientist in Certus Software Verification & Validation Center, Simula Research Laboratory, Norway. He has been affiliated to Simula Research Lab since 2007. He has been involved in many industrial and research projects related to Model-Based Testing (MBT) and Empirical Software Engineering since 2003. He has experience of working in several industries and academia research groups in many countries including UK, Canada, Norway and Pakistan. He is a member of the IEEE Computer Society.

Arnaud Gotlieb obtained his PhD degree in the fields of Software Testing and Constraint Programming from the University of Nice-Sophia Antipolis, in 2000. He has worked 7 years in the Industry at THALES (pre. DASSAULT Electronics) where he was successively PhD student, engineer and project manager in the field of Software Testing. From 2002 to 2011, he has been worked in Rennes as an INRIA's researcher on Constraint-Based Testing. Since 2011, he joined the Simula Research Laboratory as a senior research scientist and from 2012, he leads the CERTUS Software Validation & Verification Center and heads the Software Engineering department of SIMULA. He is the main author of more than forty publications and coauthor of more than seventy, and he is the main architect of several constraint solving engines targeted to the testing of critical programs. He is a member of the IEEE Computer Society.