Minimizing Test Suites in Software Product Lines Using Weight-based Genetic Algorithms

Shuai Wang
Certus Software V&V Center
Simula Research Laboratory
Dept of Informatics, University of Oslo
P.O. Box 134, Lysaker, Norway
shuai@simula.no

Shaukat Ali
Certus Software V&V Center
Simula Research Laboratory
P.O. Box 134, Lysaker, Norway
shaukat@simula.no

Arnaud Gotlieb
Certus Software V&V Center
Simula Research Laboratory
P.O. Box 134, Lysaker, Norway
arnaud@simula.no

ABSTRACT

Test minimization techniques aim at identifying and eliminating redundant test cases from test suites in order to reduce the total number of test cases to execute, thereby improving the efficiency of testing. In the context of software product line, we can save effort and cost in the selection and minimization of test cases for testing a specific product by modeling the product line. However, minimizing the test suite for a product requires addressing two potential issues: 1) the minimized test suite may not cover all test requirements compared with the original suite; 2) the minimized test suite may have less fault revealing capability than the original suite. In this paper, we apply weight-based Genetic Algorithms (GAs) to minimize the test suite for testing a product, while preserving fault detection capability and testing coverage of the original test suite. The challenge behind is to define an appropriate fitness function, which is able to preserve the coverage of complex testing criteria (e.g., Combinatorial Interaction Testing criterion). Based on the defined fitness function, we have empirically evaluated three different weight-based GAs on an industrial case study provided by Cisco Systems, Inc. Norway. We also presented our results of applying the three weight-based GAs on five existing case studies from the literature. Based on these case studies, we conclude that among the three weight-based GAs, Random-Weighted GA (RWGA) achieved significantly better performance than the other ones.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms, Performance, Experimentation, Verification

Keywords

Test minimization, feature pairwise coverage, fault detection capability, weight-based GAs

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

1. INTRODUCTION

Software Product Line Engineering (SPLE) is becoming a cost-effective way to develop similar products by exploiting and managing commonalities and variabilities among a large number of products [1][2]. SPLE 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 [2].

For testing a product line, in the current practice of industry (e.g., based on our experience of working with Cisco Systems, Inc. Norway), a test suite is typically developed to test the whole product line and the test suite will be modified as new products come into play or the current products need to be improved [1][3]. However, as the number of products increases, the number of test cases for testing the product line will also increase. Therefore, it becomes practically impossible to execute all the test cases of the product line due to limited available time and resources for each new product. It is therefore essential to seek a solution to minimize test suites for a specific product efficiently before execution to reduce the cost of testing [3].

Based on our collaboration with Cisco, a methodology has been proposed to support automated test case selection using Feature Model (FM) and Component Family Model (CFM) [4][5]. This methodology captures the commonalties and variabilities of a product line using a FM and the domain knowledge of test experts using a CFM. However, after carefully studying the results, we observed there still exist redundant test cases in the selected test suite and eliminating these redundant test cases can reduce the execution cost of testing (test minimization). Moreover, there are two potential risks for test minimization, i.e., the minimized test suite might not cover all testing functionalities (i.e., test requirements) and have lower fault detection capability than the original test suite. Therefore, we are facing a challenge, in our industrial context, to minimize the test suite for testing a product, whereas at the same time achieving high fault detection capability and functionality coverage. However, the current practice of reducing the number of test cases is manual and not systematic, which often decreases the functionality coverage and fault detection capability in addition to being time consuming as we observed in our previous work [4].

Based on the above-mentioned challenges, the problem we are targeting is a multi-objective optimization problem and multi-objective Genetic Algorithms (GAs) are known for solving such a problem [6]. Moreover, Weight-based GAs suits our context well since our goal is to minimize the test suite for testing a product, while keeping high feature pairwise coverage of testing and fault detection capability, i.e., we aim at balancing all the proposed objectives and achieving them best together. Therefore, in this paper, we propose an application of weight-based Genetic Algorithms (GAs) to minimize test suites for a product. More specifically, we first formally define the above-mentioned issues as three objectives: Test Minimization Percentage (*TMP*), Feature Pairwise Coverage (*FPC*) and Fault Detection Capability (*FDC*), and propose objective functions for them. Second, a fitness function is defined, based on these objectives to guide the search towards an optimal solution. Third, we present an industrial case study and five other case studies from SPLOT (http://www.splot-research.org/) to evaluate three different weight-based GAs (i.e., Weight–Based Genetic Algorithm (WBGA), Weight-Based Genetic Algorithm for Multi-objective Optimization (WBGA-MO), Random-Weighted Genetic Algorithm (RWGA)) based on the proposed fitness function. Random Search (RS) is used as the baseline to evaluate the performance of the selected weight-based GAs. Last, based on the obtained results and analysis, we propose a guideline for future practitioners to solve test minimization problems using weight-based GAs.

The contributions of this paper are: 1) Definition of a fitness function based on the three objectives; 2) Empirical evaluation of the three weight-based GAs by comparing them with RS, based on an industrial case study and five case studies from SPLOT.

The rest of paper is organized as follows: Section 2 provides a briefly introduction of the existing methodology for test case selection and the weight-based GAs we applied. Section 3 presents a formal representation of the problem, definitions and functions for all the three objectives followed by fitness function used by all the algorithms. In Section 4, empirical evaluation based on an industrial case study and five other case studies are presented. Section 5 discusses threats to validity and related work is presented in Section 6. Section 7 concludes the paper.

2. BACKGROUND

In this section, we briefly present our existing methodology for test case selection based on FM and CFM in the context of product line (Section 2.1) followed by related description of selected weight-based GAs (Section 2.2).

2.1 Test Case Selection based on FM and CFM

Feature modeling is a hierarchical modeling approach for capturing commonalities and variabilities in product line [2][7]. By modeling a product line with Feature Model (FM), various products can be configured through feature selection in a systematic way. A component family model (CFM) is used to represent how products are assembled and generated in a product line by modeling relations among software architectural elements [8]. It has a hierarchical structure including items such as *components* and *parts*. Meanwhile, *restrictions* play a key role, which are used to specify relations between features in FM and components/parts in CFM. Via *restrictions*, a specific product can be configured automatically from a valid selected feature model in a product line.

In our previous work [4][5], we proposed a systematic and automated methodology using a Feature Model for Testing (FM_T) to capture commonalities and variabilities of a product line and a Component Family Model for Testing (CFM_T) to capture the overall structure of test cases in the repository. With our methodology, a test engineer does not need to manually go through the repository to select a relevant set of test cases for a new product. Instead, a test engineer only needs to select a set of relevant features using FM_T at a higher level of abstraction for a product and a set of relevant test cases will be selected automatically. In addition, we developed a tool named as Import Plugin and Transformation (IPT) to support this automated test case selection methodology and it has been applied in Cisco Systems, Norway.

2.2 Description of Selected Weight-based GAs

Genetic Algorithms (GAs) are inspired by the Darwinian evolution theory, which are well known to address complex problems in Search-Based Software Engineering (SBSE) [9]. When applying GA, a population of individuals (i.e., candidate solutions) is evolved through a series of generations, where reproducing individuals evolve through crossover and mutation operators. Meanwhile, GAs are often compared with the techniques based on random search (RS) when applying to determine whether the complexity is warranted to address a specific SBSE problem. The use of the GAs may only be justified if it performs significantly better than, for instance, RS. To use a GA, a fitness function needs to be defined to evaluate the quality of a candidate solution (i.e., an element in the search space). The fitness function is problem dependent, and proper care needs to be taken for developing adequate fitness functions. The fitness function will be used to guide the search toward fitter solutions. Below, we provide a brief description of the weight-based GAs that we used in this paper, i.e., Weight–Based Genetic Algorithm (WBGA), Weight-Based Genetic Algorithm for Multi-objective Optimization (WBGA-MO), and Random-Weighted Genetic Algorithm (RWGA).

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 [6]. The difference of these algorithms can be summarized as the different mechanisms of assigning weights for each objective, which is shown in Table 1.

Table 1 Mechanisms of Assigning Weights for the Three Weight-based GAs

Algorithms	Mechanisms of Assigning Weights
WBGA	Predefined fixed weights
WBGA-MO	Weights in weight pool
RWGA	Randomly normalized weights

As shown in Table 1, WBGA assigns fixed weights for each objective defined for each objective before, which can be provided by users based on the domain knowledge and expertise. The same weight for each objective is used during all generations of the algorithm. WBGA-MO uses a weight pool including a set of predefined weights and each solution can choose weights for each objective randomly during running. The assigned weight for each objective changes during each generation, but is still limited from ones existing in the weight pool. Notice that such weight pool can also be provided by users. As for RWGA, different normalized weights are assigned randomly for each objective at each generation, i.e., it is not required to assign particular weights before and weights for each objective are assigned dynamically during running.

Based on the above discussions, these algorithms can be classified as *fixed-weight* algorithms and *dynamic-weight* algorithms according to their assigning ways of weights. Therefore, WBGA and WBGA-MO can be classified into *fixed-weight* algorithms and RWGA can be classified into *dynamic-weight* algorithms.

3. PROBLEM REPRESENTATION AND FITNESS FUNCTION

To guide the search towards an optimal solution using weight-based GAs, it is essential to define a fitness function, as is the case for any search algorithm. Our goal is to find a minimum subset of test cases to test a product in a product line from the entire set of test cases available for the product line. At the same time, we also want to achieve high feature pairwise coverage and fault detection capability. In Section 3.1, we first present formal representation of the search problem. Second, we define three objectives and related functions, i.e., *TMP*, *FPC* and *FDC* in Section 3.2 followed by the definition of the fitness function in Section 3.3.

3.1 Problem Representation

Suppose a product line *P* has a set of products $P = \{p_1, p_2, p_3 \dots p_{np}\}$, where *np* is the number of products in *P*. Moreover, *P* can be represented as a feature model with a set of features $F = \{f_1, f_2, f_3 \dots f_{nf}\}$ [2][5], where *nf* is the number of features (i.e., functionalities need to be tested for *P*). To test *P*, there is a test suite $TS = \{t_1, t_2, t_3 \dots t_{nt}\}$ comprising of a large number of test cases (*nt*). A specific product p_i in *P*, where $1 \le i \le np$, can be represented as a subset of F : $F_{p_i} = \{f_1', f_2', f_3' \dots f_{nf_{p_i}}'\}$, where f_i' can be any feature in F ($f_i' \in F$) and nf_{p_i} is the number of features used to represent the product p_i ($1 \le nf_{p_i} \le nf$). To test p_i, based on [4], there is a subset of the test suite *TS*: $TS_{p_i} = \{t_1', t_2', t_3' \dots t_{nt_{p_i}}'\}$ comprising nt_{p_i}test cases, where t_i' can be any test case in *TS* ($t_i' \in TS$) and $1 \le nt_{p_i} \le nt$. Based on TS_{p_i} for p_i, there is a set of potential solutions $S_{p_i} = \{s_{p_i1}, s_{p_i2}, s_{p_i3} \dots s_{ns_{p_i}}\}$, where $ns_{p_i}$is the total number of solutions for testing p_i, which can be measured as $2^{nt_{p_i}} - 1$. 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 [4], 100 test cases are obtained to test the product p_i. Then there will be $2^{100} - 1$ potential solutions for testing p_i, which is a huge solution space. Each solution s_i in S_{p_i} is comprised of nt_{s_i} test cases from TS_{p_i}: $\{t_{s_i1}', t_{s_i2}', t_{s_i3}' \dots t_{nt_{s_i}}'\}$, where $1 \le nt_{s_i} \le nt_{p_i}$ and has a certain FPC_{s_i} and FDC_{s_i}, which are measured as described in the next section.

Based on the above discussion, the detailed problem can be represented as:

Problem: Search for a solution s_k (s_k is comprised by $\left\{t_{s_k1}', t_{s_k2}', t_{s_k3}' \dots t_{nt_{s_k}}'\right\}$ from TS_{p_i}, where $1 \le nt_{s_k} \le nt_{p_i}$) from S_{p_i} for testing the product p_i to achieve 1) high TM_{s_k}(i.e., less number of test cases); 2) high FPC_{s_k}; 3) high FDC_{s_k}.

3.2 Definitions and Functions for Three Objectives

In this section, based on the above-mentioned problem, we provide mathematical definitions and functions for these three objectives required to achieve test minimization.

3.2.1 Test Minimization Percentage (TMP)

TMP is used to measure the amount of reduction in the number of test cases and is calculated as follows.

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

As discussed in section 3.1, $nt_{s_k}$is the number of test cases for the solution s_k, where $1 \le nt_{s_k} \le nt_{p_i}$. 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* represents higher test minimization.

3.2.2 Feature Pairwise Coverage (FPC)

FPC is used to measure how much pairwise coverage can be achieved by a chosen solution [10][11]. We chose this type of coverage based on our domain knowledge, discussion with test engineers, and history data about faults because a higher percentage of detected faults are mainly due to the interactions between features. *FPC* is designed to compute the capability of covering feature pairs by a chosen solution, which is computed as below:

$$FPC_{s_k} = \frac{Num\_FP_{s_k}}{Num\_FP_{p_i}} * 100\%$$

$Num\_FP_{s_k}$ is the number of feature pairs covered in the test cases for the solution s_k, which can be 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 $1 \le nt_{s_k} \le nt_{p_i}$. $Num\_FP_{tc_i}$ 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: $Num\_FP_{tc_i} = C_{size\,(F_{tc_i})}^2$. $size\,(F_{tc_i})$ 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 *i* are C_3^2 = 3*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 $Num\_FP_{s_k}$.

$Num\_FP_{p_i}$ is all number of feature pairs for testing the product p_i which can be measured as: $Num\_FP_{p_i} = C_{\text{size}\left(F_{p_i}\right)}^2 = nt_{p_i} * (nt_{p_i} - 1)/2$. $F_{p_i}$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 the product are C_{10}^2= 10*9/2 = 45. Note that *FPC* is calculated for a chosen test solution and ranges from 0 to 1 and a higher value of *FPC* shows higher feature pairwise coverage.

3.2.3 Fault Detection Capability (FDC)

FDC measures the fault detection capability of a selected test solution for a product. In out context, fault detection refers to the success rate of a test case in a given time, e.g., a week or a month. In our context, a test case is defined as a *success* if it can detect faults in a given time and as a *fail* if it does not detect any fault. The success rate of a test case can be measured as below.

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

$SucR_{tc_i}$ is the success rate of execution for test case *i* during the given time (in our case, the given time is per week);

$NumSuc_{tc_i}$ is the number of *success* executions for the given test case i during the given time;

$NumFail_{tc_i}$is the number of *fail* executions for the given test case *i* during the given time.

For instance, a test case is usually executed 1000 times per week in Cisco. So if a test case executes successfully for 800 times in the given week, the fault detection capability for the test case is 800/1000 = 0.8. Similarly, for a test solution including a set of test cases for a specific product, the fault detection capability can be measured as below.

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

nt_{s_k} is the number of test cases for the solution s_k, where $1 \leq nt_{s_k} \leq nt_{p_i}$. Note that *FDC* value also ranges from 0 to 1 and a higher value of *FDC* represents higher fault detection capability.

3.3 Fitness Function

In this stage, *TMP*, *FPC*, and *FDC* have been defined and can be computed through the mathematical formulas. To ease computation of the fitness function, the values for all the three objectives have been normalized by the above-proposed formulas, which range from 0 to 1. We adopted a fitness function for weight-based GAs presented in [6][12], and is defined as follows:

$$min\, Fitness\_F = 1 - (w_1 * TMP + w_2 * FPC + w_3 * FDC)$$

w_1, w_2 and w_3 are a set of assigned weights to *TMP*, *FPC* and *FDC* respectively, and they need to satisfy the constraint: $w_1 + w_2 + w_3 = 1$. 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 [6]. Notice that different set of weights can represent distinct testing preferences. For instance, if *TMP* is considered as the most important objective, w_1 can be assigned the highest weight, such as 0.6. In practice, w_1, w_2 and w_3 must be identified in a more systematic way such as by a thorough domain analysis followed by a comprehensive questionnaire. In our context, we obtained two sets of values of w_1, w_2 and w_3 via domain analysis and discussion with test engineers of Cisco, i.e., w_1 = 1/3, w_2= = 1/3, w_3= 1/3 and w_1 = 0.2, w_2 = 0.4, w_3 = 0.4.

In this paper, three different weight-based GAs are used, which are the commonly used GAs based on weight theory, i.e., WBGA, WBGA-MO, and RWGA. We have used two sets of weights for WBGA which are: WBGA_W$_1$ (W$_1$ = (1/3, 1/3, 1/3)), WBGA_W$_2$ (W$_2$ = (0.2, 0.4, 0.4). As for WBGA-MO, we create weight pool using these two sets of weights for different objectives.

4. CASE STUDIES AND EMPIRICAL EVALUATION

In this section, first we present an industrial case study and five other case studies and then an empirical evaluation of various weight-based GAs is presented.

4.1 Case Studies

We evaluated our fitness function using one industrial case study and five case studies from the literature.

4.1.1 Industrial Case Study

Our industrial case study is provided by Cisco and is part of a large project on model-based testing of a Video Conferencing System (VCS) product line called Saturn [13]. The Saturn product line comprises of several VCSs such as C20, C40, and C60. The Saturn product line has more than 2000 test cases and for each product (e.g., C20), only a subset of all test cases is needed.

We chose four products C20, C40, C60 and C90 from Saturn. Based on our domain analysis, there are 169 features in Saturn and each product includes a subset of all features. Meanwhile, each feature can be tested by at least one test case (usually more than one). Table 2 shows more details about this. For instance, C20 contains 17 features (i.e., testing functionalities) and 138 test cases are used to test these features. Each test case tc_i has a success rate for execution ($SucR_{tc_i}$). In general, for Saturn, each feature is associated with 5-10 test cases; each test case is associated with 1-5 features and the fault detection capability ranges from 50% to 95%.

Table 2 Four Products in Saturn

Product	#Features	# Test Cases
C20	17	138
C40	25	167
C60	32	192
C90	43	239

4.1.2 Other Case Studies

Our other case studies are of five other products from different product lines from SPLOT (http://www.splot-research.org/). Each product line is represented by a feature model and each product includes a subset of all features [14]. Meanwhile, all test cases for testing the product line can be represented by a component family model and a subset of test cases can be obtained automatically for testing a specific product as discussed in [4]. For other five products represented by different sets of features from SPLOT as shown in Table 3. Here, we follow the similar phenomenon observed from Saturn and assume that each feature can be tested by 5-10 test cases, each test case can be used to test 1-5 features and the success rate (*SucR*) for each test case ranges from 50% to 95%.

Table 3 Five Case Studies from SPLOT

Name	Author	Description	#Features
Car Software System	Chr Wol	Simple model of a car's software product line	18
ATM Software	TCN	A feature model for ATM software	29
DELL Laptop/Notebook Computers	Moises Branco	A feature model describes the features of DELL Laptop/Notebook Products	46
SmartHome	Conejero	Adaptation of the original feature models for the SmartHome system used by AMPLE project as case study	59
J2EE web architecture	Reinout Korbee	A feature model for web architectures	77

4.2 Empirical Evaluation

This section discusses experiment design, execution, and analysis of the evaluation based on the guidelines reported in [15][16].

4.2.1 Experiment Design

The goal of our experiments is to assess the effectiveness of weight-based GAs for test minimization and at the same time achieving high feature pairwise coverage and fault detection capability.

4.2.1.1 Research Questions

From this experiment, we want to answer the following research questions:

RQ1: Are weight-based GAs effective to solve test minimization problem in our context?

RQ2: How do the weight-based GAs (WBGA, WBGA-MO, RWGA) compare to RS and among WBGA, WBGA-MO, RWGA, which one fares best in solving test minimization problem?

4.2.1.2 Experiment Settings

In our experiments, we 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$_1$ = (1/3, 1/3, 1/3)), WBGA_W$_2$ (W$_2$ = (0.2, 0.4, 0.4)), 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. Meanwhile, the size of population and maximum number of fitness evaluation are set as 100 and 2000, respectively. Finally, RS was used as the comparison baseline to assess the difficulty of the addressed minimization problems [16]. Notice that different settings may lead to different performance for genetic algorithms, but standard settings usually perform well [15].

Moreover, a set of threshold values for *TMP*, *FPC*, and *FDC* are selected that show the minimum acceptable values for a particular context in our experiments. Note that these thresholds are set through the domain analysis and discussion with test engineers and a test manager at Cisco, and history data about test execution. In the context of our industrial case study, these thresholds values are: *TMP*:=0.8; *FPC*:=0.8; and *FDC*:=0.85.

4.2.1.3 *Statistical* Tests

To compare the obtained result and given thresholds, the Vargha and Delaney statistics and Mann-Whitney U test are used based on the guidelines for reporting statistical tests for randomized algorithms presented in [15].

- $\hat{A}_{12}$**:** The Vargha and Delaney statistics is used to calculate $\hat{A}_{12}$, which is a non-parametric effect size measure [15]. In our context, given performance measure $F = 1 - (w_1 * TMP + w_2 * FPC + w_3 * FDC)$, $\hat{A}_{12}$ is used to compare the probability of yielding higher performance value F for two algorithms A and B. If $\hat{A}_{12}$ is equal to 0.5, the two algorithms are equivalent. If $\hat{A}_{12}$ is greater than 0.5, it means the first algorithm A has higher chances of obtaining higher F value than B.
- ***p*-value**: The non-parametric U-test (The Mann-Whitney U test) is used to calculate p-value for deciding whether there is a significant difference between two algorithms. We chose the significance level of 0.05, which means there is a significant difference if p-value is less than 0.05.

Based on the above description, we define that algorithm A has better performance than algorithm B, if the $\hat{A}_{12}$ value is greater than 0.5. Moreover, algorithm A has significantly better performance than algorithm B, if the $\hat{A}_{12}$ value is greater than 0.5 and p-value is less than 0.05.

4.2.2 Experiment Execution

According to the guidelines in [15], each algorithm is run for 1000 times to account for random variations inherited in search algorithms and is essential to increase the power of statistical tests that further improves the confidence on the results. We let WBGA_W$_1$, WBGA_W$_2$, WBGA-MO, RWGA and RS run up to 2000 fitness generations each time and collected the optimal solution including the final value of fitness function. We ran our experiments on a PC with Intel Core i7 2.3GHz with 4 GB of RAM, running Microsoft Windows 7 operating system.

4.2.3 Results and Analysis

In this section, we discuss and analyze the obtained results for the individual research question followed by an overall discussion. Note that Table 4 and Table 5 are shown in the last page of paper as landscape to save the space.

4.2.3.1 Results and Analysis for Research Question 1

Table 4 shows the results, when the performance of search algorithms is compared with the threshold values for our industrial case study and five case studies from SPLOT. In Table 4, we did one sample Mann Whitney test since we compared the obtained results from different algorithms with one set of fixed values for *TMP*, *FPC* and *FDC*.

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

RWGA has higher probability to obtain better results when compared with the given thresholds. In other words, RWGA has higher probability to be adapted when required since most of $\hat{A}_{12}$ values are greater than 0.5 and there is no significant difference (all p-values are greater than 0.05) when comparing with the given thresholds as shown in Table 4.

In a way, the results obtained by WBGA-MO are almost equivalent to the given thresholds since most of $\hat{A}_{12}$ values are close to 0.5. Meanwhile, there are no significant difference between the results of WBGA-MO and the given thresholds (all p-values are greater than 0.05) as shown in Table 4.

For WBGA_W$_1$ and WBGA_W$_2$, the results do not stay stable. For some products, $\hat{A}_{12}$ values are close to 0.5, for other products, $\hat{A}_{12}$ values are much less than 0.5 (the given thresholds have much higher probability to be used). Meanwhile, for some products, there is no significant difference when compared with the given thresholds and there is a significant difference with the thresholds for other products as shown in 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 reasonable weights before using WBGA is essential to obtain 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 $\hat{A}_{12}$ values are less than 0.5 and there are significant differences when comparing with the given thresholds since p-values are less than 0.05 as shown in Table 4.

In general, weight-based GAs can assist to achieve the given thresholds. Given different weights, using WBGA can obtain total different results. So assigning a reasonable set of weights before using WBGA plays a key role to gain expected results. Notice that the obtained results by WBGA-MO and RWGA are more stable than WBGA, which is because WBGA-MO and RWGA are less dependent on a certain set of weights. Especially, since there is no need to provide weights before using, the obtained results by RWGA are more stable than WBGA-MO.

4.2.3.2 Results and Analysis for Research Question 2

Table 5 shows the results when different algorithms are compared for the Saturn products and five other products. According to the combination theory, 10 (C_5^2 = 5*4/2) pairs are compared.

Based on the obtained results, we can answer *RQ2* as follows:

Firstly, as shown in Table 5, we compared the performance of various algorithms with RS. When WBGA_W$_1$ is compared with RS, the $\hat{A}_{12}$ values are all greater than 0.5 and the *p*-values are mostly less than 0.05 (8 out of 9 in four VCS products and five other case studies). For the comparison of WBGA_W$_2$ and RS, the $\hat{A}_{12}$ values are mostly greater than 0.5 (8 out of 9 in four VCS products and five other case studies) and the *p*-values are all less than 0.05. When comparing WBGA-MO with RS, the $\hat{A}_{12}$ values are all greater than 0.5 and the *p*-values are all less than 0.05. Finally, when RWGA is compared with RS, the $\hat{A}_{12}$ values are all greater than 0.5 and the *p*-values are all less than 0.05. Based on the above results, we can conclude that weight-based GAs have significantly better performance than RS.

Now we discuss the results, when various weight-based GAs are compared. Firstly, when WBGA_W$_1$ is compared with RWGA, Table 5 shows that the $\hat{A}_{12}$ values are all less than 0.5 and the *p*-values are mostly less than 0.05 (8 out of 9 in four VCS products and five other case studies); secondly, comparing WBGA_W$_2$ with RWGA, the $\hat{A}_{12}$ values are all less than 0.5 and the *p*-values are also mostly less than 0.05 (8 out of 9 in four VCS products and five other case studies); thirdly, comparing WBGA-MO with RWGA, the $\hat{A}_{12}$ values are all less than 0.5 and the *p*-values are all less than 0.05. Based on the above results, we concluded that RWGA has significantly better performance than WBGA_W$_1$, WBGA_W$_2$ and WBGA-MO.

Finally, WBGA_W$_1$, WBGA_W$_2$ and WBGA-MO were compared. Since both WBGA_W$_1$ and WBGA_W$_2$ belong to WBGA, we studied them together and compare WBGA and WBGA-MO. Based on the obtained results as shown in Table 5, for the $\widehat{A}_{12}$ values, 10 out of 18 are less than 0.5 and 8 out of 18 are greater than 0.5 in four VCS products and five other case studies, and for the *p*-values, 4 out of 18 are less than 0.05 and 14 out of 18 are greater than 0.05. Based on the results, we can conclude that WBGA-MO has better performance than WBGA, but not significantly.

4.2.3.3 Overall Discussion

Based on the above-mentioned discussions when using weight-based GAs, different sets of weights (e.g., w_1, w_2 and w_3) can be provided depending on the testing requirements. So domain knowledge and thorough discussions with users is required before assigning weights.

Moreover, based on the obtained results and analysis, RWGA has the best performance among the three weight-based GAs and RS. The main reason is that RWGA assigns weights dynamically during search and thus the search is guided towards the best weights to achieve threshold values for *TMP*, *FPC* and *FDC*.

In particular, when testing requirements are not clear in practice, i.e., it is impossible to provide reasonable weights for *TMP, FPC* and *FDC*. We suggest using RWGA to minimize test suites for products since it assigns weights dynamically. In the context of VCS testing, after running RWGA 1000 times, based on the empirical evaluation, the best set of weights that obtains the best results of fitness function is w_1 = 0.24 for *TMP*, w_2 = 0.44 for *FPC* and w_3 = 0.32 for *FDC* respectively.

5. THREATS TO VALIDITY

To reduce *construct validity* threats, we used the same stopping criteria for all algorithms, i.e., number of fitness evaluations. We ran each algorithm for 2000 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.

A possible threat to *internal validity* is that we have experimented with only one configuration setting for the GA parameters. However, these settings are in accordance with the common guidelines in the literature and our previous experience on testing problems. Parameter tuning can improve the performance of GAs, although default parameters often provide reasonable results [15].

We ran our experiments on an industrial case study to seek the best solution to minimize test suites for testing a product. To reduce *external validity* threats (i.e., our results might not be applicable to other empirical studies), five other case studies from SPLOT were adapted using the same criteria, i.e., each feature can be tested by 5-10 test cases, each test case can be used to test 1-5 features and success rate for each test case ranges from 50% to 95%. In this way, the results obtained by the industrial case study and five other case studies should be more persuasive.

The most probable conclusion validity threat in experiments involving randomized algorithms is due to random variations. To address it, we repeated experiments 1000 times to reduce the possibility that the results were obtained by chance. Furthermore, to determine the probability of yielding higher performance by different algorithms, we measured the effect size using $\hat{A}_{12}$ by Vargha and Delaney statistics test. We chose Vargha and Delaney statistics test since it is appropriate for non-parametric effect size measure, which match our situation [15]. Meanwhile, we performed Mann-Whitney U test to determine the statistical significance of the results.

Finally, as already discussed in Section 4, our practical set of weights obtained by empirical evaluation may not be applicable to all product line testing. Therefore, in every specific product line needs to be tested, a different set of weights might be obtained when using RWGA. However, for some specific product line similar to VCS product line, the proposed set of weights could be adapted directly.

6. RELATED WORK

Software product line testing is a relatively new, but intense field of research since product line engineering has shown significant benefits in both academia and industry [1][17][18].

In [1], McGregor presented a set of activities, which can be used to address testing individual assets (e.g., verification of consistency between requirements and specifications) and testing artifacts (e.g., test-case derivation and test suite design) that represent complete products in the context of product line. In [17], based on a relational model capturing variability in product line, Cohen defined a family of cumulative coverage criteria to collect test coverage information, which was also used to map with existing combinatorial testing approaches for supporting product line testing. In [18], Muccini proposed associating regression testing with product line by comparing code execution with the architectural design, which can be used as guidelines for adapting existing techniques for regression testing to product line testing. However, most of these works are only proposals and only provide guidelines without specific testing process, which are not systematic. In our previous work [4], an methodology was presented to obtain a set of test cases for testing a specific product automatically by modeling software product line using feature model and test case structure using component family model, which is more related with test selection and there are still redundant test cases in the obtained test suite for a specific product through more investigation.

As for test minimization in regression testing, various types of minimizing techniques are proposed in the literatures [19]. In [20], Chen and Lau proposed GE and GRE heuristics to perform test minimization, which are thought as the variation of the greedy algorithms. In [21], Tallam and Gupta proposed an improved greedy algorithm called *the delayed greedy* approach based on the Formal Concept Analysis of the relation between test cases and testing requirements. In [22], Jeffrey and Gupta extended GE and GRE by introducing a secondary set of test requirements to determine whether a test case is redundant for test minimization. Although the number of techniques for regression test minimization is huge [19], there is no enough evidence to prove that these techniques still work well if being adapted in the context of product line.

In particular, Shin and Mark [23] proposed to use search-based algorithms (i.e., Fast Non-dominated Sorting Genetic Algorithm (NSGA II)) for multi-objective test case selection (e.g., code coverage, fault detection history, execution time) in regression testing, which is similar with our two objectives (i.e., *FPC* and *FDC*). However, our context is product line and we minimized the test suite obtained by our proposed test case selection methodology in [4] for testing a product. Moreover, we defined one additional objective (i.e., *TMP* to measure the minimization percentage of the minimized test suite as compared with the original test suite based on our industrial problem. In addition, we compared different weight-based GAs and RS, which was not addressed in [23].

7. CONCLUSION AND FUTURE WORK

In this paper, we proposed an application of weight-based GAs to minimize the test suite for testing a product at the same time achieving high feature pairwise coverage and fault detection capability in the context of software product line. We formally defined three objectives (i.e., Test Minimization Percentage (*TMP*), Feature Pairwise Coverage (*FPC*) and Fault Detection Capability (*FDC*)), followed by the definition of a fitness function (based on the objectives) to guide three different weight-based Genetic Algorithms (GAs): Weight–Based Genetic Algorithm (WBGA), Weight-Based Genetic Algorithm for Multi-objective Optimization (WBGA-MO), Random-Weighted Genetic Algorithm (RWGA).

We evaluated our fitness function based on an industrial case study and five other case studies from the literature using the three weight-based GAs. Given a set of thresholds, these three weight-based GAs and Random Search (RS) were evaluated and a comparison among them was conducted.

Results show that RWGA and WBGA-MO achieved the given thresholds, WBGA achieved the given thresholds in some products and failed in others, and RS hardly achieved the given thresholds. Among all of these algorithms, RWGA has the best performance, and WBGA-MO and WBGA are almost equivalent, both of which have better performance than RS.

In the future, we plan to conduct more case studies using the proposed application of weight-based GAs. Moreover, we want to investigate the quality of the minimized test suite as compared with the test suite selected by test engineers manually to assess whether the weight-based GAs are human competitive. Meanwhile, we will also evaluate the proposed fitness function using Pareto-based multi-objective GAs, such as Fast Non-dominated Sorting Genetic Algorithm (NSGA II), to validate whether better results can be obtained as compared to weight-based GAs.

ACKNOWLEDGMENTS

The work reported in this paper is funded by the Norwegian Research Council under the research-based innovation scheme (SFI) in the Certus Center hosted by Simula Research Laboratory. We would like to thank Marius Christian Liaaen (Cisco Systems, Inc. Norway) for providing us the detailed case study and thorough discussion.

8. REFERENCES

[1] J. McGregor. Testing a Software Product Line. Technical Report CMU/SEI-2001-TR-022. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania. 2001. http://www.sei.cmu.edu/library/abstracts/reports/01tr022.cfm

[2] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems.* (35), 615–636. 2010.

[3] T. Chen, and M. Lau. Dividing strategies for the optimization of a test suite. *Information Processing Letters*. 60(3), pp. 135– 141. 1996.

[4] S. Wang, A. Gotlieb, M. Liaaen, and L. C. Briand. Automatic selection of test execution plans from a Video Conference System Product Line. In *Proceedings of the ACM MODELS Workshop VARiability for You (VARY' 12*), pp. 30-35. 2012.

[5] S. Wang, A. Gotlieb, S. Ali, M. Liaaen. Automated Selection of Test Cases using Feature Model: An Industrial Case Study. Technical Report (2012-20), Simula Research Laboratory. 2012.

[6] A. Konak, D. W. Coit, and A. E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*. 91(9), pp. 992-1007. 2007.

[7] K. Czarnecki, C. Kim, and K. Kalleberg. Feature models are views on ontologies. In *Proceedings of International Software Product Line Conference*, pp. 41–51. 2006.

[8] Pure systems GmbH. Variant management with pure::variants. Technical white paper. Available from http://web.pure- systems.com, 2006.

[9] M. Harman, S. A. Mansouri, and Y. Zhang. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. Technical Report TR-09-032009. King's College, London. 2009.

[10] K. C. Tai, and Y. Lei. A Test-Generation Strategy for Pairwise Testing. *IEEE Trans. of Software Engineering*. 28(1), pp. 109 - 111. 2002.

[11] R. Kuhn, Y. Lei, and R. Kacker. Practical combinatorial testing: Beyond pairwise. *IT Professional*. 10(3), pp. 19-23. 2008.

[12] T. Murata, T. Ishibuchi, and H. Tanaka. Multi-objective genetic algorithm and its applications to flowshop scheduling. *Comput Ind Eng*. 30(4), pp. 957–968. 1996.

[13] Cisco Systems. Cisco telepresence codec c90. Data sheet. 2010. Available from http://www.cisco.com.

[14] D. Benavides. On the Automated Analysis of Software Product Lines Using Feature Models. Doctoral Thesis. Universidad de Sevilla. 2007.

[15] A. Arcuri, and L. C. Briand. A Practical Guide for Using Statistical Tests to Assess Randomized Algorithms in Software Engineering. In *Proceedings of the International Conference on Software Engineering*. pp. 21-28. 2011.

[16] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege. A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation. *IEEE Trans on Software Engineering*, 36(6), pp. 742-762. 2010.

[17] M. B. Cohen, M. B. Dwyer, and J. Shi. Coverage and Adequacy in Software Product Line Testing. In *Proceedings ACM ISSTA Workshop on Role of Software Architecture for Testing and Analysis*. pp. 53-63. 2006.

[18] H. Muccini, and A. Van Der Hoek. Towards Testing Product Line Architectures. *Electronic Notes in Theoretical Computer Science*. 82(6), pp. 99-109. 2003.

[19] S. Yoo, and M. Harman. Regression test minimization, selection and prioritization: a survey. *Software Testing, Verification, and Reliability*; 22(2), pp. 67-120. 2012.

[20] T. Y. Chen, and M. F. Lau. Dividing strategies for the optimization of a test suite. *Information Processing Letters*. 60(3), pp. 135–141. 1996.

[21] S. Tallam, and N. Gupta. A concept analysis inspired greedy algorithm for test suite minimization. *SIGSOFT Software Engineering Notes*. 31(1), 35–42. 2006.

[22] D. Jeffrey, and N. Gupta. Test suite reduction with selective redundancy. In *Proceedings of the International Conference on Software Maintenance*. pp. 549-558. 2005.

[23] S. Yoo, and M. Harman. Pareto Efficient Multi-Objective Test Case Selection. In *Proceedings of the international symposium on Software testing and analysis (ISSTA),* pp. 140-150.2007.

Table 4 Results for Comparing Different Algorithms with the Given Thresholds*

PA	C20		C40		C60		C90		S1		S2		S3		S4		S5	
	A	p	A	p	A	p	A	p	A	p	A	p	A	p	A	p	A	p
C1	0.34	**0.04**	0.29	**0.03**	0.22	**0.01**	0.41	**0.08**	0.41	**0.08**	0.34	**0.04**	0.55	0.16	0.44	0.24	0.23	**0.01**
C2	0.42	**0.06**	0.51	**0.04**	0.54	**0.09**	0.45	0.33	0.37	**0.04**	0.29	**0.02**	0.43	0.44	0.38	**0.09**	0.35	0.12
C3	0.55	0.23	0.48	0.19	0.61	0.12	0.38	**0.06**	0.44	**0.05**	0.41	0.22	0.64	**0.08**	0.46	0.15	0.39	**0.07**
C4	0.71	0.35	0.62	0.22	0.67	0.46	0.66	0.18	0.53	0.26	0.49	0.60	0.59	**0.07**	0.57	0.45	0.45	0.32
C5	0.39	**0.01**	0.42	**0.01**	0.22	**0.01**	0.18	**0.03**	0.28	**0.02**	0.22	**0.01**	0.41	0.29	0.32	**0.01**	0.29	**0.05**

*PA: Pair of algorithms, A: $\hat{A}_{12}$, p: *p*-value
S1: Car Software System, S2: ATM Software, S3: DELL Laptop/Notebook Computers, S4: SmartHome, S5: J2EE web architecture.
C1: WBGA_W_1 vs. Thresholds, C2: WBGA_W_2 vs. Thresholds, C3: WBGA-MO vs. Thresholds, C4: RWGA vs. Thresholds, C5: RS vs. Thresholds.
All *p*-values less than 0.05 are identified as bold.

Table 5 Results for Comparing Different Algorithms*

PA	C20		C40		C60		C90		S1		S2		S3		S4		S5	
	A	p	A	p	A	p	A	p	A	p	A	p	A	p	A	p	A	p
C6	0.48	0.15	0.52	0.25	0.41	0.51	0.55	0.10	0.51	0.61	0.55	0.59	0.47	0.15	0.43	0.35	0.49	0.25
C7	0.38	0.35	0.48	**0.03**	0.47	0.57	0.52	0.72	0.48	0.09	0.41	0.15	0.39	0.35	0.52	0.12	0.54	0.12
C8	0.33	**0.02**	0.45	0.42	0.22	**0.01**	0.29	**0.03**	0.29	**0.02**	0.31	**0.03**	0.22	**0.01**	0.27	**0.01**	0.33	**0.04**
C9	0.62	**0.06**	0.70	**0.01**	0.55	0.14	0.66	**0.02**	0.62	**0.02**	0.69	**0.04**	0.73	**0.01**	0.57	**0.02**	0.78	**0.01**
C10	0.51	0.15	0.44	0.61	0.39	0.09	0.53	0.52	0.44	0.52	0.51	0.35	0.57	0.09	0.46	0.27	0.55	0.16
C11	0.31	**0.03**	0.49	**0.05**	0.32	**0.02**	0.42	0.42	0.21	**0.02**	0.33	0.05	0.18	**0.01**	0.26	**0.02**	0.22	**0.03**
C12	0.69	**0.01**	0.58	**0.02**	0.62	**0.01**	0.48	**0.04**	0.55	**0.04**	0.64	**0.01**	0.51	0.05	0.73	**0.01**	0.59	**0.02**
C13	0.37	0.16	0.31	**0.03**	0.29	**0.01**	0.27	**0.01**	0.49	0.19	0.38	**0.03**	0.29	**0.01**	0.32	**0.02**	0.41	**0.01**
C14	0.59	**0.01**	0.65	**0.01**	0.61	**0.01**	0.56	**0.01**	0.68	**0.02**	0.59	**0.02**	0.62	**0.02**	0.71	**0.01**	0.56	**0.03**
C15	0.74	**0.02**	0.69	**0.01**	0.77	**0.03**	0.62	**0.02**	0.81	**0.02**	0.67	**0.01**	0.74	**0.01**	0.89	**0.01**	0.65	**0.02**

*C6: WBGA_W_1 vs. WBGA_W_2, C7: WBGA_W_1 vs. WBGA-MO, C8: WBGA_W_1 vs. RWGA, C9: WBGA_W_1 vs. RS, C10: WBGA_W_2 vs. WBGA-MO, C11: WBGA_W_2 vs. RWGA, C12: WBGA_W_2 vs. RS, C13: WBGA -MO vs. RWGA, C14: WBGA-MO vs. RS, C15: RWGA vs. RS.
All *p*-values less than 0.05 are identified as bold.