# Exploring Test Suite Diversification and Code Coverage in Multi-Objective Test Case Selection

Debajyoti Mondal, Hadi Hemmati, and Stephane Durocher
Department of Computer Science, University of Manitoba, Winnipeg, MB, Canada
Email: {jyoti,hemmati,durocher}@cs.umanitoba.ca

***Abstract*—Test case selection is a classic testing technique to choose a subset of existing test cases for execution, due to the limited budget and tight deadlines. While 'code coverage' is the state of practice among test case selection heuristics, recent literature has shown that 'test case diversity' is also a very promising approach. In this paper, we first compare these two heuristics for test case selection in several real-world case studies (Apache Ant, Derby, JBoss, NanoXML and Math). The results show that neither of the two techniques completely dominates the other, but they can potentially be complementary. Therefore, we next propose a novel approach that maximizes both code coverage and diversity among the selected test cases using NSGA-II multi-objective optimization, and the results show a significant improvement in fault detection rate. Specifically, sometimes this novel approach detects up to 16%(Ant), 10%(JBoss), and 14%(Math) more faults compared to either of coverage or diversity-based approaches, when the testing budget is less than 20% of the entire test suite execution cost.**

## I. Introduction

The prime objective in software testing is to maximize the number of faults detected within a limited testing budget. Test case selection is one of the most classic problems in this context. For many years, code coverage has been the only practical approach for test case selection. Recent studies, however, reveals that code coverage may not necessarily be strongly correlated with test suite effectiveness [1]. Consequently, several other heuristics, e.g., selecting test cases depending on their past fault coverage history, eliminating test cases with similar coverage and so on, have been introduced to improve the test case selection effectiveness. Diversity-based test case selection is one of these new approaches with promising results [2]. The core idea behind the diversity-based approach is to select the most diverse subset of test cases rather than focusing on maximizing code coverage. Given the assumption that there is no evidence on the location of source code faults before testing, all parts (in any given granularity, e.g., file, method, or statement-level) of the system under test (SUT) are equally fault-prone. Therefore, a diversity-based test case selection optimizes the selected test cases by making sure that they cover all parts evenly – we do not want to spend all testing budget on a small part of the code-base, if there is no evidence that the concentrated part is more fault-prone than other parts.

In this paper, we explore the effectiveness of coverage-based and diversity-based test case selection, using their fault detection ability for a given testing budget. We measure testing budget as test execution time and apply a bi-objective optimization algorithm to maximize the coverage/diversity, while minimizing test execution time. Since the existing literature finds combination of Additional-Greedy and NSGA-II is a good choice for optimizing coverage and diversity over time [3], [2], we also use these algorithms for the bi-objective optimization.

Then we compare the fault detection rate of the two test case selection approaches on sixteen versions of five open source projects (Apache Ant: 1 version, Derby: 1 version, JBoss: 3 versions, NanoXML: 3 versions, and Math: 8 versions). The results show that for some versions of Ant, JBoss, NanoXML and Math, diversity-based approach is significantly more effective in revealing faults than coverage. On the other hand, for Derby and some other versions of Math, the fault detection rate for coverage is significantly larger than that of diversity. Hence we examined whether the solutions obtained by diversity and coverage-based heuristics are complementary, and realized that the two solution sets (the subsets given by coverage-based and diversity-based test case selection approaches) have little overlap. Therefore, we suggest a novel three-objective optimization approach using NSGA-II to maximize both code coverage and diversity, while minimizing test execution time. Afterwards, we analyze the results of an experiment on the same sixteen SUTs and compare the effectiveness of the novel three-objective approach with the two bi-objective approaches. The results show that optimizing both heuristics together improves the fault detection rate of the selected test cases. For example, there exists cases where the combined approach improves the bi-objective approach by up to 16%(Ant V7), 10%(JBoss V1), and 14%(Math V105) when selecting test cases with execution cost equal to 20% or less than the entire test suite's cost. Besides, we could not find any scenario where the three-objective approach is significantly worse than both bi-objective formulations, which suggests that the three-objective formulation is also a safe technique.

*Contributions:* The main contributions of this paper are listed below.

A. The paper proposes a new approach for bi-objective optimization of diversity and test execution time, using $\alpha$-Shape analysis of the Pareto front solutions.
B. The paper compares the effectiveness of diversity-based vs. coverage-based test case selection, in terms of fault-detection rate, on sixteen versions of five real-world programs (Apache Ant: 1 version, Derby: 1 version, JBoss: 3 versions, NanoXML: 3 versions, and Math: 8 versions).
C. The paper proposes a novel three-objective test case selection approach that maximizes code coverage and test suite diversity, while minimizing the test execution time.
D. The paper also empirically evaluates the proposed approach and compares that with the two bi-objective approaches (diversity-time and coverage-time). The results show sig-

nificant improvement in terms of fault-detection rate on the selected test cases from the SUTs under investigation.

## II. Background

In this section we introduce a brief overview of test case selection heuristics, algorithms and evaluation techniques.

### A. Test Case Selection

Over the last decade, test case selection has been an important task in different software testing phases. For example, the test case selection problem appears while we construct a test suite from a set of automatically generated test cases, or in regression testing, where we select a subset of test cases from a test suite to test the newer version of a program. Although the goal in the test case selection problem is to choose those test cases that would reveal the maximum number of faults, aiming for such an optimum selection is impractical since the faults are not known beforehand. Hence a common strategy is to select those test cases that have the maximum amount of code coverage [3], with the hope that covering more code helps revealing more faults. A more recent heuristic is to select test cases that are dissimilar to each other [2], [4]. Some other test case selection heuristics optimize the total execution time or past fault coverage. In the following, we describe the coverage- and diversity-based test case selection techniques in more detail.

*1) Coverage-based Test Case Selection:* The most widely examined objective for test case selection is code coverage [5], [6]. The code coverage of a test case can be measured at various levels of details. The coverage of a test case can measured by the percentage of total the number of lines executed by the test case. However, such information is usually unavailable unless the code is executed. Determining the coverage without executing the test cases requires static analysis of the test case, where a test case can be encoded by its associated function calls [7]. The coverage of such a test case is the percentage of total functions that it calls. For example, the following test case fragment can be encoded as ⟨`StringUtils.split, StringUtils.lineSplit, StringUtils.replace`⟩. If the number of distinct functions in the system under test is 100, then the coverage of the fragment is 3%.

```
public void testStringUtils(){
    final String data = "a,b,,";
    Vector res = StringUtils.split(data,
        ',');
    assertEquals(4, res.size());
    assertEquals("a", res.elementAt(0));
    final String data = "a\r\nb\nc\nd\ne";
    Vector res =
        StringUtils.lineSplit(data);
    assertEquals(5, res.size());
    assertEquals("c", res.elementAt(2));
    final String data = "abcabcabca";
    String res = StringUtils.replace(data,
        "a", "");
    assertEquals("bcbcbc", res);
}
```

*2) Diversity-based Test Case Selection:* Over the past few years, test case diversity is shown to be an important optimization function in test case selection [2], [4], [8]. Intuitively, diversity between two test cases is a distance function that measures their dissimilarity. The diversity of a set of three or more test cases is the average pairwise diversity of that set.

Assume that each test case is encoded as a binary vector, where 1s correspond to the program units such as functions that are called in the test case. For such an encoding, diversity functions used in the literature [2], [9] are Hamming distance, Dice diversity, Levenshtein, and so on.

**Hamming diversity:** Hamming distance is a distance measure on two sequences of bits of equal length. Hamming distance between two test cases is the ratio of the number of mismatches over the total number of positions. For example, let $A = \langle 1, 1, 0, 1, 0, 0 \rangle$ and $B = \langle 1, 0, 1, 1, 0, 1 \rangle$ be two test cases, then $Hamming\_div\ (A, B) = 3/6 = 0.5$.

**Levenshtein:** Levenshtein diversity measures the dissimilarity based on the string edit distance [2], [10]. Unlike Hamming distance, this does not require the sequences (i.e., test cases) to have equal length. Levenshtein distance is the minimum number of edit operations (insertion, deletion, or replacement) needed to transform one sequence into another. For example, the Levenshtein distance bettween the test cases $A = \langle f_1, f_3, f_7, f_8, f_9 \rangle$ and $B = \langle f_3, f_4, f_5, f_8 \rangle$ is four since one can delete $f_1$ from $A$ and then replace $f_7, f_8, f_9$ by $f_4, f_5, f_8$ to obtain the target sequence $B$.

**Dice diversity:** Hemmati et al. [4] found the Gower-Legendre (or *Dice*) formula to be most effective in model-based testing. Given two test cases $A$ and $B$, each represented by a binary vector with ones denoting the function calls associated with the corresponding test case, the diversity between $A$ and $B$ is

$$div(A, B) = 1 - \frac{|A \cap B|}{|A \cap B| + w(|A \cup B| - |A \cap B|)}$$

where $w = 0.5$ for the Dice function. If $A = \langle 1, 1, 0, 1, 0, 0 \rangle$ and $B = \langle 1, 0, 1, 1, 0, 1 \rangle$, then $Dice\_div\ (A, B) = 0.43$.

**Diversity vs. Coverage:** Coverage and diversity are two fundamentally different criteria. While the coverage never decrease with the increase in the number of selected test cases, the diversity value may decrease. Besides, a set of test cases with the maximum coverage does not guarantee that the set has fair amount of diversity and vice versa. As an example consider the following three test cases $T_1, T_2, T_3$.

```
T1(){
  a.f1();
  a.f2();
  a.f3();
  a.f4();
}
```

```
T2(){
  a.f1();
  a.f2();
  a.f5();

}
```

```
T3(){
  a.f6();


}
```

Table I shows the encoding of these test cases, and the (Dice) diversity and coverage values for all possible subsets of two test cases. The set $\{T_1, T_2\}$ achieves maximum possible coverage with any two test cases, but does not have maximum diversity. On the other hand, the set $\{T_2, T_3\}$ maximizes diversity, but cannot achieve the maximum coverage. Note that

although $\{T_1, T_2\}$ achieves maximum coverage, the test cases $T_1$ and $T_2$ are very similar, and hence they may reveal the same fault. However, maximizing both coverage and diversity gives the set $\{T_1, T_3\}$ that achieves the same amount of coverage as $\{T_1, T_2\}$, but the dissimilarity of $T_1$ and $T_3$ makes them more likely to detect different faults.

TABLE I. (LEFT) TEST-CASE ENCODING. (RIGHT) COVERAGE AND DIVERSITY ACHIEVED FOR DIFFERENT PAIRWISE SELECTION.

| | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ |
|---|---|---|---|---|---|---|
| $T_1$ | 1 | 1 | 1 | 1 | 0 | 0 |
| $T_2$ | 1 | 1 | 0 | 0 | 1 | 0 |
| $T_3$ | 0 | 0 | 0 | 0 | 0 | 1 |

| | Cov. | Div. |
|---|---|---|
| $T_1, T_2$ | 5/6 | 3/7 |
| $T_1, T_3$ | 5/6 | 1 |
| $T_2, T_3$ | 4/6 | 1 |

### B. Optimization Algorithms

The problem of selecting a set of $k$ test cases, for some fixed value of $k$, that achieves maximum coverage or diversity is NP-complete [11], [12]. Hence researchers have studied several greedy and search based optimization techniques for test case selection. Here we briefly describe some of these algorithmic techniques.

*1) Single-objective Optimization:* In a single-objective test case selection, the objective of an algorithm is to maximize a single objective function, i.e., either coverage or diversity.

**Greedy and Additional-Greedy Algorithms:** The Greedy algorithm starts with an empty solution set and in each iteration, it chooses the best test case among the available test cases. For example, a greedy ordering of the test cases in Table I is $T_1, T_2, T_3$ (optimizing coverage). In a test case selection problem, a sequence of test cases are selected from the beginning of this sequence depending on the available budget and resources for test execution.

A variant of the Greedy algorithm is the Additional-Greedy algorithm. Here the next test case is chosen such that it gives the best objective value while considered along with all the selected test cases. While optimizing coverage, the two valid Additional-Greedy orderings of the test cases in Table I are $T_1, T_2, T_3$ and $T_1, T_3, T_2$. Similarly, while optimizing Dice diversity, the valid Additional-Greedy orderings are $T_2, T_3, T_1$ and $T_1, T_3, T_2$. Note that the Additional-Greedy algorithm for diversity starts with the most diverse pair of test cases, while for coverage the algorithm starts with a single test case with the highest coverage. Additional-Greedy has been found to be more effective than Greedy [3] for optimization problems.

**Genetic Algorithms:** Genetic Algorithms [13] (GAs) are search algorithms based on the mechanism of natural selection and natural genetics. GAs start off by an initial population of individuals (each of them corresponds to a set of test cases, i.e., a candidate for an optimal solution) and improves the quality of individuals (in terms of the objective value, i.e., coverage or diversity) by iterating through generations. The transaction from one generation to another consists of four main operations: selection, crossover, mutation and sampling.

**Others:** Ramanathan et al. [14] proposed a graph-based algorithm for the test case ordering, where each node of the graph is a distinct test case and each edge of the graph is weighted by the dissimilarity between the corresponding pair of test cases. They used Fiedler (spectral) ordering of the nodes to find the prioritization order. Their experimental evaluation suggested that the average performance of Fiedler ordering and greedy-based ordering is similar, but Fiedler ordering performs better in the worst case. Some other optimization test case selection algorithms [2], [15] are base on integer programming, clustering, hill climbing, simulated annealing, and so on.

*2) Multi-objective Optimization:* A multi-objective test case selection algorithm seeks to simultaneously optimize multiple objective functions at the same time. For example, in a regression test case selection, one may want to find a solution that has good code coverage and does not take much time to execute. In such a scenario, the trade-off among the optimization criteria becomes non-trivial. Instead of arguing whether a set of test cases with $(coverage, executiontime) = (40\%, 0.5h)$ is better than another set of test cases with $(coverage, executiontime) = (60\%, 1h)$, a multi-objective test case selection reports both solutions. Specifically, we report the set of all non-dominated solutions, i.e., the Pareto front. In the following we describe the multi-objective test case selection techniques in details.

*3) Pareto Front:* Pareto optimality is a concept used frequently in economics and game theory for multi-objective optimization. In game theory, Pareto efficiency is said to be a strategy in which one player's situation cannot be improved without making the other player's situation worse. In terms of multi-objective optimization, Pareto optimality is defined as follows. Given a vector of $M$ objective functions $f_i(x)$, where $i = 1, 2, \ldots, M$, the objective is to find a decision vector $x$ of variables that optimizes the functions vector. A decision vector $x$ *dominates* a decision vector $y$ if and only if the followings are satisfied:

$$f_i(x) \geq f_i(y), \forall i = 1, 2, \ldots, M, \text{and}$$
$$\exists i \in \{1, 2, \ldots, M\} | f_i(x) > f_i(y).$$

While the above equation is for maximization, the analogous notion holds also for the minimization problem. Consider now the set $S$ of decision vectors such that no vector in $S$ is dominated by any other decision vector. Then the vectors in $S$ are *Pareto optimal* and the corresponding objective functions $f_i(x)$ form the *Pareto front*. Figure 1(left) illustrates a Pareto front. Figure 1(right) further illustrates the non-dominating property of the Pareto optimal solutions, i.e., solution $b$ is dominated by $a$ since $a$ achieves higher coverage spending a smaller amount of execution time. However, $a$ and $c$ are both on the Pareto front, and hence none of them is dominated by the other.

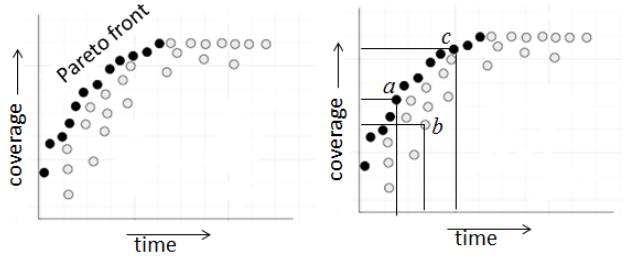


Fig. 1. (left) The points on the Pareto front are shown in black. (right) Illustration for domination and non-domination.

*4) $\alpha$-Shape:* A Pareto front analysis may sometimes provide us little information about the best solution vectors if the objective function value does not change monotonically over time. We observed such a scenario while working with diversity, as in Figure 2(left). Unlike coverage, the increase in diversity with time is initially very sharp, but starts to decrease soon. However, over time diversity may again increase and then decrease. Due to the sharp initial increase in diversity, the Pareto front of the diversity-time plot contains only a few points (e.g., the points enclosed in squares in Figure 2(left)). Hence to select the optimal choices over time, we try to determine the shape of the plot. Specifically, we choose the points on the upper envelope of the $\alpha$-shape of the diversity-time plot, i.e., the black points in Figure 2(left).

An *$\alpha$-shape* is a mathematically well defined geometric concept [16] that captures the shape of a point set, where the parameter $\alpha$ determines the curvature in the shape. A more formal definition is as follows. Let $P$ be a set of points in the Euclidean space. For $\alpha > 0$, let $D_\alpha$ be a disk of radius $1/\alpha$ that does not contain any point of $P$. Then the *$\alpha$-shape* of $P$ is the complement of the union of all $D_\alpha$s. Figure 2(right) illustrates the $\alpha$-shape of the diversity-time plot along with the disks. Note that before computing the $\alpha$-shape we add some dummy points along the time-axis. Hence the upper envelope of the $\alpha$-shape corresponds to the optimal solutions over time. Since we are only interested in the upper envelope of the $\alpha$-shape, throughout the paper, while we refer to an $\alpha$-shape, we only consider the upper envelope of the shape.

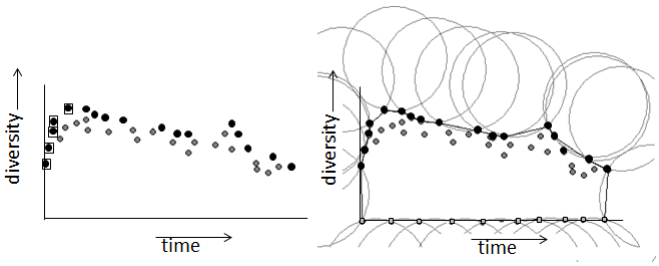


Fig. 2. (left) The points on the $\alpha$-shape and Pareto front are shown in black and enclosed in square, respectively. (right) Illustrating $\alpha$-shape using disks.

*5) Optimization Techniques:* The algorithms for single-objective optimization is also applicable to the multi-objective optimization problem. However, in this case we need to combine multiple objectives into a single objective function. A traditional approach to combine coverage and execution time is to take the ratio of coverage and time as a single objective [3]. Similarly, we can combine diversity and execution time by taking their ratio as the objective function. For more than two objective functions, a simple strategy is to optimize the weighted average of the objective values [3].

**NSGA-II Algorithm:** The Non-Dominating Sorting Genetic Algorithm (NSGA-II) is a genetic algorithm-based optimization technique developed by Deb et al. [17] for multi-objective optimization. The output of NSGA-II is a set of non-dominated solutions. Besides, NSGA-II tries to spread the solution all over the Pareto front. Deb et al.'s implementation of NSGA-II is capable of handling constraints. For example, if the constraint is to select a set with at least 10 test cases, then every solution generated by NSGA-II contains at least 10 test cases.

## III. Empirical Studies

In this section we present the experimental details, discuss the research questions and evaluation techniques.

### A. Experimental design

*1) Systems Under Test:* We study the following programs in this paper:

- **Apache Ant:** This Java program is a Java-based build tool, similar to the Unix tool 'make', which consists of ≈80500 LOC and 627 classes.
- **Derby:** Derby is a Java-based database management system, which consists of around 503833 LOC and 1967 classes.
- **JBoss:** JBoss is a Java-based application platform for web services, which consists of 116638 LOC and 1125 classes.
- **NanoXML:** NanoXML is a XML parser for Java, which consists of around 7646 LOC and 24 classes.
- **Math:** Commons Math is an open-source Java-based library for statistics and mathematics, which consists of around 85000 LOC.

The above programs are all real-world applications. We chose these systems to be both diverse in terms of application domain and be publicly available (The dataset for Math versions can be found from [18] and the rests at Software-artifact Infrastructure Repository (SIR) [19]). We also chose SUTs with both seeded and real faults, to make sure we are not biased toward only seeded faults, which are more common in academic public repositories.

Table II shows the number of test cases, the number of method calls that appeared in all test cases, and the fault status of different programs.

TABLE II. Fault Status.

| System | Tests | Method calls | Faults | Avg. faults per test |
|---|---|---|---|---|
| Ant V7 | 105 | 320 | 6 (seeded) | 0.21 |
| Derby V5 | 52 | 258 | 26 (seeded) | 5.09 |
| JBoss V1 | 97 | 422 | 10 (seeded) | 0.91 |
| JBoss V2 | 147 | 621 | 35 (seeded) | 4.20 |
| JBoss V3 | 133 | 564 | 9 (seeded) | 3.35 |
| NanoXML V1 | 74 | 26 | 7 (seeded) | 0.45 |
| NanoXML V2 | 74 | 23 | 7 (seeded) | 0.13 |
| NanoXML V3 | 76 | 23 | 7 (seeded) | 0.15 |
| Math V99 | 146 | 528 | 5 (real) | 0.03 |
| Math V100 | 146 | 528 | 6 (real) | 0.04 |
| Math V101 | 146 | 528 | 6 (real) | 0.04 |
| Math V102 | 144 | 512 | 10 (real) | 0.06 |
| Math V103 | 140 | 495 | 8 (real) | 0.05 |
| Math V104 | 140 | 495 | 9 (real) | 0.06 |
| Math V105 | 100 | 414 | 7 (real) | 0.07 |
| Math V106 | 97 | 414 | 8 (real) | 0.08 |

*2) Test Case Representaion and Execution Time:* We represent a test case by a sequence of method calls and thus the diversity and coverage will be defined on the method call levels. We analyzed each test case statically to extract its method calls (similar to [20], [7]).

Note that for our analysis, we need the execution time of the test case. We estimated the execution time of a test case in a test suite as the percentage of total method calls in the test suite that appeared in the test cases.

### B. Research Questions

Since we are comparing the effectiveness of diversity and coverage in revealing faults, we ask the following.

**RQ1.** Which test case selection heuristic, among diversity-time and coverage-time based formulations, is more effective in revealing faults?

**RQ2** Are the solutions produced by diversity-time and coverage-time formulations complementary?

Diversity and Coverage are completely different approaches toward test selection. Therefore, a follow up question on RQ1, *i.e.*, RQ2, is how much overlap the two solutions have and if little, can we improve both by combining them, which is examined in RQ3?

**RQ3.** Does three-objective formulation improve fault detection rate of the bi-objective formulations, from RQ1?

### C. Execution of Algorithms and Evaluation

We used the Additional-Greedy and NSGA-II algorithms for bi-objective formulations (i.e., for optimizing coverage over time and diversity over time), as the most effective algorithms reported in the literature [3], [2].

For NSGA-II, we used a random initial population of size 200. We iterated the algorithm for 5000 fitness evaluations, with single point crossover and bit-flip mutation. We stopped at 5000 since our tuning showed that running NSGA-II for extended period of time does not show any noticeable improvement in its performance.

For three-objective formulation (i.e., when optimizing both diversity and coverage over time), we computed a global Pareto frontier from the solutions found by NSGA-II. To have a fair comparison, in this case we also used a random initial population of size 200, with 5000 fitness evaluations with single point crossover and bit-flip mutation.

To take into account the inherent randomness of the algorithms, for each system under test, we executed 20 independent runs of these algorithms. Note that the running time for Additional-Greedy is negligible compared to the running time of NSGA-II. For example, while a single run for Apache Ant using NSGA-II (for bi-objective formulation) takes around 2 minutes, Additional-Greedy takes around 10 seconds. For three-objective formulation, NSGA-II takes around 5 minutes.

## IV. Results

In this section we present and analyze the results of our experiments, and discuss the research questions based on the empirical analysis.

**Discussion on RQ1.** RQ1 asks about the relative effectiveness of the bi-objective optimizations, in terms of fault detection.

For a fair comparison among these two approaches, we need to use the algorithms that best optimize the diversity and coverage. Let X and Y be the techniques that best optimize diversity and coverage, respectively. We then assess the solutions produced by X and Y and identify the more effective approach.

Previous studies [3], [2] have shown that the Additional-Greedy and NSGA-II algorithms are very successful for both optimizing coverage over time and diversity over time. Therefore, we used these algorithms for our experiments.

While optimizing coverage over time, we computed a global Pareto frontier from the solutions found by both algorithms. Similarly, while optimizing diversity over time, we computed a global $\alpha$-shape from the solutions found by these algorithms. Besides, to have a good spread of the solutions over the Pareto front and $\alpha$-shape, we executed NSGA-II several times with the constraint that each solution must have at least a particular number of test cases.

Note that by executing the algorithms, we obtain two sets of solutions. One of these sets is computed by optimizing diversity over time, and the other set is computed by optimizing coverage over time, while both sets are constructed using the best algorithms (Additional-Greedy and NSGA-II) suggested in the literature. We now examine the fault-revealing capabilities of these two sets of solutions, which helps us to understand the relative effectiveness of the diversity-time and coverage-time formulations.

Figure 3 illustrates the mean percentage of faults detected over time, where the error bars correspond to the 95% confidence interval. Since in the context of test case selection the smaller subsets are usually preferred (due to limited budget), we look into the results for the data over the first 20% of total test suite's execution time.

For eleven of the sixteen systems under study (Ant V7, JBoss V1-2, NanoXML V1-3, Math V99-104), as the time increases, diversity-based approach achieves higher mean faults than the coverage-based approach. For some Math versions (e.g., Math V99-104), diversity continues to detect newer faults while coverage seems to saturate over time.

However, there are scenarios where the performance of diversity is not very distinctive compared to the performance of coverage-based approach, and sometimes coverage seems to perform even better than diversity-based approach. For example, the effectiveness of diversity and coverage seems similar for JBoss V1 and V3. On the other hand, for Derby V5 and Math V105-106, the number of faults detected using coverage-time formulation are much larger than the faults revealed by the diversity-time formulation.

Table III shows the same results in more details. The term $c_i$ denotes the median of the percentage of faults revealed, while using the coverage-based approach within the time interval $[0, t_i]$. Similarly, $d_i$ denotes the median of percentage of faults revealed while using diversity-based approach within the time interval $[0, t_i]$.

We conducted Mann-Whitney U test on these data, to see whether the differences are statistically significant. The cells where the difference in medians is statistically significant are shaded with light-green or dark-blue depending on whether diversity or coverage has the higher median. Note from the table that the faults revealed by diversity over execution time is up to 28% more (e.g., NanoXML V1-3, Math V102, Math V104) than the percentage of faults detected by coverage. The scenario is different for Derby and some versions of Math,

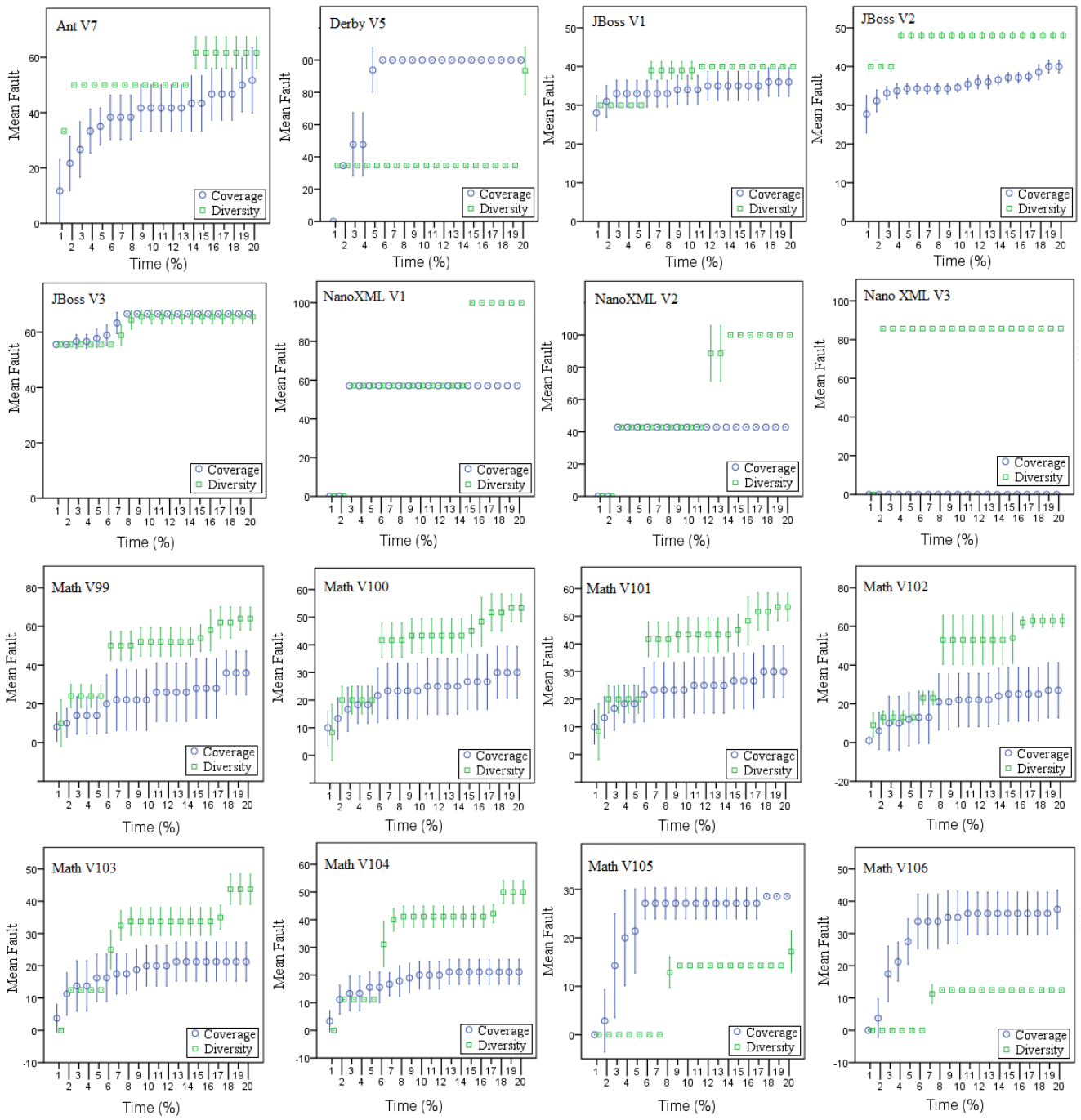


Fig. 3. Mean percentage of faults detected over time, where time is shown in percentage of total execution time of all the test cases, in the test suite. The error bars correspond to the 95% confidence interval.

where sometimes coverage detects up to 15% (Math V105-106) more faults than diversity. And there are cases like JBoss V3, where the performances of diversity and coverage do not show any statistical significance.

In summary, our experimental findings show that though diversity appears to be more effective in more cases, none of the diversity and coverage-based approaches completely dominates the other. Hence for the better understanding the relation among the two sets of solutions, we examine, in RQ2, whether they contain complementary information.

**Discussion on RQ2.** To examine whether solutions obtained using diversity and coverage-based heuristics are complementary, we examined how many solutions are common among all the solutions computed using diversity and coverage. Note that by the solutions generated using the diversity and coverage-based approach we refer to the solution points on the $\alpha$-shape, and on the Pareto front, respectively.

Figure 4(left) illustrates that only a few solutions (at most 3% of all the solutions generated) are common to both heuristics. Another interesting observation is that the

TABLE III. MEDIAN PERCENTAGE OF FAULTS REVEALED OVER TIME (FOR 20 INDEPENDENT RUNS OF THE ALGORITHMS). THE CELLS WHERE THE DIFFERENCE IN MEDIANS IS STATISTICALLY SIGNIFICANT ARE SHADED WITH LIGHT-GREEN OR DARK-BLUE DEPENDING ON WHETHER DIVERSITY OR COVERAGE HAS THE HIGHER MEDIAN.

| | $t_i(\%)$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ant V7 | $d_i$ | 33 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 67 | 67 | 67 | 67 | 67 | 67 | 67 |
| | $c_i$ | 0 | 25 | 33 | 33 | 33 | 33 | 33 | 33 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| Derby V5 | $d_i$ | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 35 | 100 |
| | $c_i$ | 0 | 35 | 35 | 35 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| JBoss V1 | $d_i$ | 30 | 30 | 30 | 30 | 30 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| | $c_i$ | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 35 | 35 | 35 | 35 | 35 | 35 | 40 | 40 | 40 |
| JBoss V2 | $d_i$ | 40 | 40 | 40 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 |
| | $c_i$ | 27 | 33 | 34 | 34 | 34 | 34 | 34 | 34 | 34 | 34 | 36 | 37 | 37 | 37 | 37 | 37 | 37 | 37 | 40 | 40 |
| JBoss V3 | $d_i$ | 56 | 56 | 56 | 56 | 56 | 56 | 56 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 |
| | $c_i$ | 56 | 56 | 56 | 56 | 56 | 56 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 |
| NanoXML V1 | $d_i$ | 0 | 0 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 100 | 100 | 100 | 100 | 100 | 100 |
| | $c_i$ | 0 | 0 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 |
| NanoXML V2 | $d_i$ | 0 | 0 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| | $c_i$ | 0 | 0 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 |
| NanoXML V3 | $d_i$ | 0 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 |
| | $c_i$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Math V99 | $d_i$ | 0 | 20 | 20 | 20 | 20 | 50 | 50 | 50 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 |
| | $c_i$ | 0 | 10 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 30 | 30 | 30 | 30 | 40 | 40 | 40 | 40 | 40 | 40 |
| Math V100 | $d_i$ | 0 | 17 | 17 | 17 | 17 | 42 | 42 | 42 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| | $c_i$ | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 25 | 25 | 25 | 25 | 33 | 33 | 33 | 33 | 33 | 33 |
| Math V101 | $d_i$ | 0 | 17 | 17 | 17 | 17 | 42 | 42 | 42 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| | $c_i$ | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 25 | 25 | 25 | 25 | 33 | 33 | 33 | 33 | 33 | 33 |
| Math V102 | $d_i$ | 10 | 10 | 10 | 10 | 10 | 20 | 20 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 |
| | $c_i$ | 0 | 0 | 0 | 0 | 0 | 5 | 5 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 30 | 30 |
| Math V103 | $d_i$ | 0 | 13 | 13 | 13 | 13 | 25 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 44 | 44 | 44 |
| | $c_i$ | 0 | 13 | 13 | 13 | 19 | 19 | 19 | 19 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 |
| Math V104 | $d_i$ | 0 | 11 | 11 | 11 | 11 | 33 | 44 | 44 | 44 | 44 | 44 | 44 | 44 | 44 | 44 | 44 | 44 | 50 | 50 | 50 |
| | $c_i$ | 0 | 11 | 11 | 11 | 17 | 17 | 17 | 17 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 | 22 |
| Math V105 | $d_i$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 | 14 |
| | $c_i$ | 0 | 0 | 14 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 29 |
| Math V106 | $d_i$ | 0 | 0 | 0 | 0 | 0 | 0 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 |
| | $c_i$ | 0 | 0 | 19 | 25 | 25 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 |

number of solutions generated by diversity-based approach is significantly smaller ($p < 0.001$) than that of coverage for every system under test. In other words, the number of solutions on the $\alpha$-shape is much smaller than the number of solutions on the Pareto front. Although this would not affect our current analysis, it is interesting to be further explored when comparing diversity and coverage heuristics.

Although there are only a few shared solutions, the two sets of solutions may still be similar. Hence we examined the similarity between solutions generated by diversity and coverage heuristics as follows: we first create all solution pairs where one solution is generated by the diversity based approach (on the $\alpha$-shape of the diversity-time plot) and one generated by the coverage-based approach (on the Pareto front of the coverage-time plot). Then we look at their similarity in terms of their relative Hamming distances. The higher Hamming distance between two solutions in such a pair suggests that the corresponding solutions are more dissimilar, and thus combining the two heuristics may be beneficial.

Figure 4(right) shows a bar plot of the Hamming distances for all solution pairs (combined over all 16 systems under study). Each bar summarizes the percentages of total pairs, where their Hamming distance is in the given range (*e.g.*, bar 10 is for pairs with $0 \leq$ Hamming distance $\leq 10$). A cumulative sum of these percentages are also depicted in the figure. As the figure shows, a very small portion of solution pairs are similar. For instance, only around 5% of the pairs are 90% similar (with Hamming distance $\leq 10$). In fact, for the majority of the cases (solution pairs), the difference in Hamming distance is around 30% to 50%. Such a dissimilarity between the solutions obtained using diversity and coverage motivated us to study the three-objective formulation that optimizes diversity, coverage and execution time, together.

**Discussion on RQ3.** RQ3 studies whether the three-objective optimization can improve the fault detection rate of the two bi-objective formulations.

We examine the faults detected by the points on the Pareto front of coverage-diversity-time plot, and then compare their fault detection rate with the bi-objective formulations. Similar to the analysis in RQ1, we look into the data over the initial 20% of the total execution time. Table IV shows the results for the systems under test. Each cell corresponds to the median of percentage of faults revealed, while using three-objective formulation, within the time interval $[0, t_i]$. We conducted Mann-Whitney U test on this data as well. The cells where the medians for three-objective formulation are significantly different and smaller than the corresponding medians for bi-objective formulation are shaded. The light-green shading on the bottom region of the cells means diversity performs significantly better than the three-objective formulation and the dark-blue on the top region of the cell means coverage performs significantly better than the three-objective formulation.

Since only a few cells of Table IV are shaded, in most of the cases the three-objective formulation is more effective in revealing faults than the two bi-objective formulations. The improvement over diversity-time formulation may reach around 16% for Ant (V7), 10% for JBoss (V1) and 6% for Math (V104-106). Similarly, the improvement over coverage-time formulation may reach around 50% for Ant (V7), 10% for JBoss (V1-2) and 11% for Math versions.

Furthermore, since none of the cells are fully shaded, there does not exist any case where the three-objective formulation is

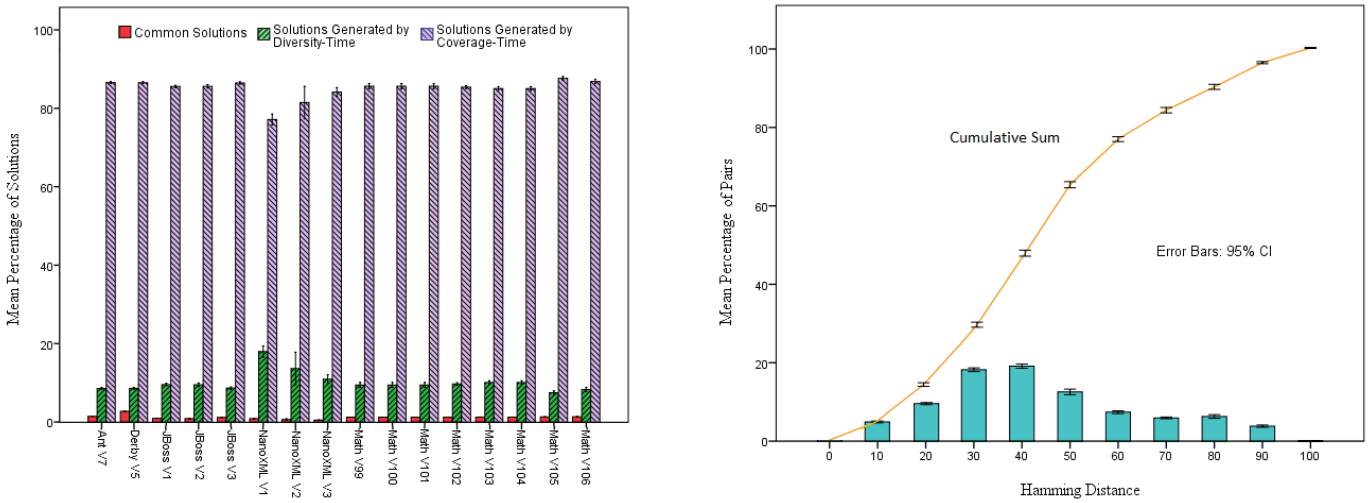


Fig. 4. (left) Mean number of solutions that are generated by diversity and coverage heuristics (classified as solutions that are generated only by each heuristics and the common solutions), for different systems under test (for 20 independent runs of the algorithms). The error bars correspond to the 95% confidence interval. (right) Percentage of the total solution pairs over the pairs' Hamming distance, depicted as both actual (bar plot) and cumulative (line plot).

TABLE IV. MEDIAN PERCENTAGE OF FAULTS REVEALED OVER TIME USING THREE-OBJECTIVE FORMULATION (FOR 20 INDEPENDENT RUNS OF THE ALGORITHMS). THE CELLS WHERE THE MEDIANS FOR THREE-OBJECTIVE FORMULATION ARE SIGNIFICANTLY DIFFERENT AND SMALLER THAN THE CORRESPONDING MEDIANS FOR BI-OBJECTIVE FORMULATION ARE SHADED. LIGHT-GREEN (ON BOTTOM REGION OF THE CELL), IF THE DIVERSITY PERFORMS BETTER THAN THE THREE-OBJECTIVE FORMULATION AND DARK-BLUE (ON TOP REGION OF THE CELL), IF COVERAGE PERFORMS BETTER THAN THE THREE-OBJECTIVE FORMULATION.

| $t_i$ (%) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ant V7 | 33 | 33 | 33 | 33 | 33 | 42 | 50 | 50 | 50 | 58 | 58 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 83 | 83 |
| Derby V5 | 0 | 35 | 35 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| JBoss V1 | 30 | 30 | 30 | 40 | 40 | 40 | 40 | 40 | 40 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| JBoss V2 | 36 | 39 | 40 | 46 | 47 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 50 | 50 | 51 | 51 | 51 | 51 | 51 |
| JBoss V3 | 56 | 56 | 56 | 61 | 61 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 | 67 |
| NanoXML V1 | 0 | 0 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 | 57 |
| NanoXML V2 | 0 | 0 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 | 43 |
| NanoXML V3 | 0 | 0 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 | 86 |
| Math V99 | 20 | 20 | 20 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 50 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 |
| Math V100 | 17 | 17 | 17 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 42 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| Math V101 | 17 | 17 | 17 | 33 | 33 | 33 | 33 | 33 | 33 | 33 | 42 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| Math V102 | 0 | 0 | 0 | 15 | 15 | 20 | 20 | 30 | 30 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 50 | 50 |
| Math V103 | 13 | 13 | 13 | 13 | 25 | 25 | 25 | 25 | 25 | 25 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 38 |
| Math V104 | 11 | 11 | 11 | 17 | 22 | 22 | 22 | 22 | 28 | 33 | 33 | 33 | 33 | 39 | 39 | 39 | 39 | 39 | 39 | 44 |
| Math V105 | 0 | 0 | 0 | 7 | 14 | 21 | 21 | 29 | 29 | 29 | 29 | 29 | 29 | 29 | 36 | 43 | 43 | 43 | 43 | 43 |
| Math V106 | 0 | 0 | 13 | 13 | 13 | 25 | 25 | 25 | 25 | 25 | 38 | 38 | 38 | 38 | 38 | 38 | 38 | 50 | 50 | 50 |

worse than both bi-objective formulations. In other words, the three-objective formulation is a *safer* approach (never being outperformed by both of the two bi-objective formulations).

The cases where three-objective formulation has higher median faults than both bi-objective formulations are particularly interesting (Ant V7, JBoss V1-2, Math V10-106). For example, looking at Table III one would observe that diversity performs better for Ant V7, coverage performs better for Math V105-106, and both performs similarly for JBoss V1. Table IV reveals that for all of these systems, the median faults for three-objective formulation is at least 10% larger than the median faults detected by each bi-objective formulation.

Note that sometimes three-objective formulation performs poorly compared to diversity-based bi-objective formulation for $t_i$ larger than 10% (e.g., NanoXML V1-2, Math V102). One plausible explanation is that the Pareto-front in the diversity-coverage-time plot favours the points having high coverage. Since we did not conduct an $\alpha$-shape analysis on the solutions obtained by three-objective approach, the three-objective formulation did not get the full benefit of choosing points that have large diversity but not Pareto optimal.

In summary, three-objective formulation is a safer approach than both diversity and coverage, which in some cases may outperform both as well.

### A. Threats to Validity

While measuring the execution time of the test cases, we estimated the execution time of a test case as the percentage of functions that appeared in the test case. Instead, one could use the CPU time taken to run the test cases. Since CPU time may

not be very precise because of internal optimization performed by CPU cache, Yoo and Harman [3] used the Valgrind profiling tool to make such computation more precise.

Another threat to our work is the optimization techniques that we used in our heuristics. Although we tried to choose the ones that are known to be the more effective ones in the literature, there is no guarantee that these algorithms are the best for our optimization. Similar argument also applies to the dissimilarity function that we used to measure diversity. Although we have examined sixteen versions of five real-world SUTs, confirming our observations requires further experiments on a larger set of programs with rich fault sets.

Finally, the effectiveness of a coverage-based approach may benefit from dynamic execution profile from history, if the coverage information is available. In this paper we did not assume availability of such information and only applied a static analysis to calculate the coverage.

## V. Related Work

Over the last decade, several test case selection techniques have been developed in the literature. Let $P$ be a program with a test suite $T$ associated with it and let $P'$ a modified version of $P$. Rothermel et al. [21] introduced the concept of selecting *modification traversing* test cases (also known as *safe selection* [22]), i.e., those test cases of $T$ that executed the code deleted from $P$, or execute the new or modified code of $P'$. A rich body of literature focuses on selecting modification traversing test cases using different techniques such as integer programming [15], data-flow analysis [23], walking in control-flow graphs [24], and so on. In some recent surveys, Yoo and Harman [25], and Biswas et al. [26] compile elaborate discussions on these techniques.

Besides safe selection of test cases, researchers have examined many other objectives and constraints for test case selection, e.g., maximizing code coverage, test case diversity, and past fault coverage, while minimizing test execution time. The most widely examined objective is code coverage [5], [6]. Leon and Podgurski [27] compared coverage-based selection with techniques using distribution of test execution profiles, where the distribution based selection was more efficient for revealing faults. Kim and Porter [28] utilized the execution history of the test cases, and suggested that historical information may be cost-effective for long-running regression testing processes. Yoo and Harman [3] proposed a multi-objective test case selection using the notion of Pareto optimality. Although they considered several objectives such as code coverage, past fault coverage and execution time, the main objective of their study was not to compare their fault detection capability, but to justify the effectiveness of Pareto-efficient analysis in optimization for multi-objective test case selection. Recently, Mei et al. [20] examined JUnit test case prioritization techniques that analyze static call graphs of the test cases and the program under test to estimate the ability of each test case to achieve code coverage.

Test case diversity is a more recently used objective function for test case selection. Tsai et al. [8] proposed a test selection technique by eliminating test cases that have similar coverage. Vega et al. [29] assessed test quality by measuring the variance of test data based on different distance metrics. Through an empirical evaluation, Hemmati et al. [4] showed that a diverse test suite is more effective in detecting faults in model based testing. In a subsequent paper, Hemmati et al. [2] examined combination of eight similarity functions (e.g., Hamming distance, Gower-Legendre, Levenshtein, and so on) and ten optimization algorithms (e.g., greedy based, clustering based, hill climbing, evolutionary algorithms and so on), where the Gower-Legendre measure for diversity and evolutionary algorithm based selection were proved to be the most successful combination. Similar to the model-based test case selection, many diversity-based selection and prioritization techniques have been proposed that do not use coverage information. For example, Ledru et al. [10] suggested a prioritization method based on the string distances between the texts of test cases. Thomas et al. [7] considered black-box test case prioritization, while modelling test cases using linguistic data (e.g., identifier names, comments, and string literals). Recently, Rogstad et al. [9] have found diversity-based test case selection to be successful to support black box regression testing of database applications.

Integer programming, greedy-based and evolutionary algorithms are some common techniques for test case selection [2], [15], [25]. Hemmati et al. [2] found (1+1)Evolutionary algorithm (with Gower-Legendre distance measure) to be the most successful test case selection technique among greedy, clustering-based or hill climbing algorithms in terms of fault detection rate. Rogstad et al. [9] also reported the success of (1+1)Evolutionary algorithm (with Mahalanobis distance measure) over greedy selection. For Pareto-efficient test case selection, Yoo and Harman [25] compared Additional-Greedy algorithm with NSGA-II [17], which is a popular multi-objective optimization algorithm. They found different scenarios where these two algorithms outperform each other.

To the best of our knowledge, optimizing diversity and coverage together, for test case selection has not been studied in the literature. The only work that considers both factors together is the work by Fraser and Wotawa [30] on test case generation. They formulated test case generation as a model checking problem that ensures that the test cases satisfying the same test requirement are actually satisfying the requirements in different ways. Unlike their work, in this paper we focus on test selection problem and define diversity on the code-level. So we do not depend on formally specified requirements.

## VI. Conclusion

Identifying the most effective set of test cases among a large test suite is one of the main challenges in software testing. In many situations, e.g., overnight regression testing, the testing time and resources are limited, which makes the execution of the entire test suite almost impossible. Several heuristics have been suggested, in the literature, for identifying the effective test cases. Among them code coverage of the test cases is the state of practice. In recent years, researchers have proposed other heuristics such as diversification as an alternative to test coverage. In this paper, we have empirically analyzed both coverage-based and diversity-based test case selection approaches in a bi-objective optimization scenario, where the first objective is maximizing diversity/coverage and the second objective is minimizing test execution time. The ultimate goal is finding the most fault revealing test cases

with minimum execution time (cost measure). Our study shows that diversity-based approach is slightly more effective than the coverage-based approach. More importantly, the study reveals that the two solution sets are barely overlapping. This motivates a three-objective approach that maximizes both diversity and coverage, while minimizing the execution time. The results show that the combined approach is significantly more effective, e.g., sometimes up to 16%(Ant V7), 10%(JBoss V1), and 14%(Math V105).

Test diversification research field is still very young and requires more theoretical and empirical studies. In the future, we plan to extend this study by analyzing additional distance functions and encodings. We are also interested in applying the historical knowledge about fault locations in the source code into the diversity function, so that the historically more fault-prone areas are assigned greater weights during test case selection. In addition, the differences between seeded and real faults are interesting to study, we leave these experiments for future work.

## ACKNOWLEDGEMENT

The research of the first author is supported in part by a University of Manitoba Graduate Fellowship, and the research of the third author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

## REFERENCES

[1] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2014, p. To appear.

[2] H. Hemmati, A. Arcuri, and L. C. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 1, pp. 1–42, 2013.

[3] S. Yoo and M. Harman, "Pareto efficient multi-objective test case selection," in *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 2007, pp. 140–150.

[4] H. Hemmati, A. Arcuri, and L. C. Briand, "Empirical investigation of the effects of test suite properties on similarity-based test case selection," in *IEEE Fourth International Conference on Software Testing, Verification and Validation (ICST)*. IEEE Computer Society, 2011, pp. 327–336.

[5] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Trans. Software Eng.*, vol. 28, no. 2, pp. 159–182, 2002.

[6] S. G. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky, "Selecting a cost-effective test case prioritization technique," *Software Quality Journal*, vol. 12, no. 3, pp. 185–210, 2004.

[7] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, "Static test case prioritization using topic," *Empirical Software Engineering*, pp. 1–31, 2012.

[8] W. T. Tsai, X. Zhou, R. A. Paul, Y. Chen, and X. Bai, "A coverage relationship model for test case selection and ranking for multi-version software," in *Proceedings of the 10th IEEE High Assurance Systems Engineering Symposium (HASE)*, 2007, pp. 105–112.

[9] E. Rogstad, L. C. Briand, and R. Torkar, "Test case selection for black-box regression testing of database applications," *Information & Software Technology*, vol. 55, no. 10, pp. 1781–1795, 2013.

[10] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," *Automotive Software Engineering*, vol. 19, no. 1, pp. 65–95, 2012.

[11] M. Gray and D. S. Johnson, "Computers and intractability: A guide to the theory of NP-completeness," *W.H. Freeman and Company*, 1979.

[12] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi, "Facility dispersion problems: Heuristics and special cases," in *Proceedings of the 2nd International Workshop on Algorithms and Data Structures (WADS)*, ser. LNCS, vol. 519. Springer, 1991, pp. 355–366.

[13] D. E. Goldberg, "Genetic algorithms in search, optimization and machine learning," 1989.

[14] M. K. Ramanathan, M. Koyutürk, A. Grama, and S. Jagannathan, "PHALANX: a graph-theoretic framework for test case prioritization," in *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC)*. ACM, 2008, pp. 667–673.

[15] K. Fischer, F. Raji, and A. Chruscicki, "A methodology for retesting modified software," in *Proceedings of the National Telecommunications Conference*, IEEE Computer Society Press: Silver Spring, 1981, pp. 1–6.

[16] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel, "On the shape of a set of points in the plane," *IEEE Transactions on Information Theory*, vol. 29, no. 4, pp. 551–558, 1983.

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

[18] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing?" in *Proceedings of the International Symposium on the Foundations of Software Engineering (FSE)*. ACM, 2009, p. To appear.

[19] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.

[20] H. Mei, D. Hao, L. Zhang, L. Zhang, J. Zhou, and G. Rothermel, "A static approach to prioritizing junit test cases," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1258–1275, 2012.

[21] G. Rothermel, M. Harrold, J. Ronne, and C. Hong, "Empirical studies of test suite reduction," *Software Testing, Verification, and Reliability*, vol. 4, no. 2, pp. 219–249, 2002.

[22] G. Rothermel and M. Harrold, "A safe, efficient algorithm for regression test selection," in *Proceedings of the International Conference on Software Maintenance (ICSM)*, IEEE Computer Press: Silver Spring, 1993, pp. 358–367.

[23] M. J. Harrold and M. Soffa, "An incremental approach to unit testing during maintenance," in *Proceedings of the International Conference on Software Maintenance (ICSM)*, IEEE Computer Society Press: Silver Spring, 1998, pp. 362–367.

[24] J. Zhao, T. Xie, and N. Li, "Towards regression test selection for aspect-oriented programs," in *Proceedings of the 2nd Workshop on Testing Aspect-oriented Programs (WTAOP)*, ACM Press: New York, 2006, pp. 21–26.

[25] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification & Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[26] S. Biswas, R. Mall, M. Satpathy, and S. Sukumaran, "Regression test selection techniques: A survey," *Informatica (Slovenia)*, vol. 35, no. 3, pp. 289–321, 2011.

[27] D. Leon and A. Podgurski, "A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases," in *Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE )*. IEEE Computer Society, 2003, pp. 442–456.

[28] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the 22rd International Conference on Software Engineering (ICSE)*. ACM, 2002, pp. 119–129.

[29] D. Vega, I. Schieferdecker, and G. Din, "Test data variance as a test quality measure: Exemplified for TTCN-3," in *Proceedings of the 19th IFIP International Conference on Testing of Communication Systems and 7th International Workshop on Formal Approaches to Testing of Software (TestCom/FATES)*, ser. Lecture Notes in Computer Science, vol. 4581. Springer, 2007, pp. 351–364.

[30] G. Fraser and F. Wotawa, "Increasing diversity in coverage test suites using model checking," in *QSIC*. IEEE Computer Society, 2009, pp. 211–218.