Test Case Selection for Black-Box Regression Testing of Database Applications

Erik Rogstad[a,b], Lionel Briand[c], Richard Torkar[d,e]

[a] *Simula Research Laboratory, Oslo, Norway*
[b] *University of Oslo, Oslo, Norway*
[c] *SnT Centre, University of Luxembourg, Luxembourg*
[d] *Chalmers University of Technology and University of Gothenburg, Göteborg, Sweden*
[e] *Blekinge Institute of Technology, Karlskrona, Sweden*

Abstract

Context: This paper presents an approach for selecting regression test cases in the context of large-scale, database applications. We focus on a black-box (specification-based) approach, relying on classification tree models to model the input domain of the system under test (SUT), in order to obtain a more practical and scalable solution. We perform an industrial case study where the SUT is a large database application in Norway's tax department.

Objective: We investigate the use of similarity-based test case selection for supporting black box regression testing of database applications. We have developed a practical approach and tool (DART) for functional black-box regression testing of database applications. In order to make the regression test approach scalable for large database applications, we needed a test case selection strategy that reduces the test execution costs and analysis effort. We used classification tree models to partition the input domain of the SUT in order to then select test cases. Rather than selecting test cases at random from each partition, we incorporated a similarity-based test case selection, hypothesizing that it would yield a higher fault detection rate.

Method: An experiment was conducted to determine which similarity-based selection algorithm was the most suitable in selecting test cases in large regression test suites, and whether similarity-based selection was a worthwhile and practical alternative to simpler solutions.

Results: The results show that combining similarity measurement with partition-based test case selection, by using similarity-based test case selection within each partition, can provide improved fault detection rates over simpler solutions when specific conditions are met regarding the partitions.

Conclusions: Under the conditions present in the experiment the improvements were marginal. However, a detailed analysis concludes that the similarity-based selection strategy should be applied when a large number of test cases are contained in each partition and there is significant variability within partitions. If these conditions are not present, incorporating similarity measures is not worthwhile, since the gain is negligible over a random selection within each partition.

Keywords: Test case selection, Regression testing, Database applications, Similarity measures

1. Introduction

Regression testing is known to be a very expensive activity as, in most cases, regression test suites are large since they attempt to exercise the system under test in a comprehensive manner. This problem is more particularly acute when test results must be manually checked, running test cases is expensive, or when access to a test infrastructure is required. As a result, as reported in a very abundant research literature [1], regression test cases must be carefully selected or prioritized. Most selection strategies are based on the static analysis of source code structure and changes. However, in many situations, this is not practical (lack of proper tool support) or applicable (no direct access to the source code or third party components) and a black-box approach, based on the system specifications, must be adopted.

In this paper, we investigate strategies for selecting regression test cases based on classification tree models. Such models have been traditionally used to partition the input domain of the system being tested [2], which in turn is used to select and generate system test cases so as to achieve certain strategies for partition coverage. Such models are

widely applied for black-box system testing for database applications and is therefore a natural and practical choice in our context. In other words, the goal is to minimize regression testing effort while retaining maximum fault detection power, and do so by relying on a specific model of the input domain.

More specifically, we combine similarity-based test case selection with such a partition-based approach to refine regression test selection. Similarity-based test case selection is a strategy that has shown itself to be cost-effective in other contexts [3, 4]. At a high level, our strategy consists in selecting regression test cases within partitions in order to maximize their diversity. A large scale experiment was conducted in the context of an industrial database application. The main goal was to investigate the impact of our proposed selection strategy in terms of fault detection rates, and the conditions under which it is beneficial. The contribution of this paper lies in defining a practical strategy for applying similarity measurements when selecting test cases generated from classification tree models, and in evaluating the approach to support regression testing in an industrial setting. This setting, further described below, is a large and business critical database application developed by the tax department of Norway.

Given the results presented in this paper, we recommend the use of similarity partition-based test case selection when the classification tree model is defined in such a way as to contain significant variability within the partitions, and the partitions generally contain many test cases. Under these conditions, it can be highly beneficial to use similarity measures as a means of selecting test cases within each partition of the classification tree model, as it leads to significant increases in fault detection rates.

The remainder of the paper is organized as follows: Section 2 describes the industrial context and the challenge addressed in the study. Section 3 provides background information regarding similarity-based test case selection and elaborates on similarity functions applicable for our context. Our proposed solution for selecting test cases generated from classification tree models, by incorporating similarity measurements, is presented in Section 4, whereas the results of the conducted industrial case study are shown in Section 5, along with a discussion of the implications of the outcomes. Conclusions are provided in Section 6.

2. Industrial setting and problem formulation

SOFIE is the tax accounting system of Norway, handling yearly tax revenues of approximately $90 billion. It has evolved over the past 10 years to provide dependable, automated, efficient and integrated services to all 430 tax municipalities and more than 3,000 end users (e.g., taxation officers). SOFIE is still evolving to accommodate changes in the tax laws, and its maintenance currently involves more than 50 employees.

The system was mainly designed to handle large amounts of data, thus requiring high throughput and a continuous focus on performance-related aspects. Historical data for all taxpayers in Norway has to be kept for at least ten years, and some of the system tables currently hold more than 500 million rows of data. To ensure efficient data processing, the business logic of the system was organized into batch jobs, along with graphical user interfaces to drive the work-processes of the end users. Both system components are tightly coupled with the API of the underlying database management system. SOFIE has approximately 380 batch jobs constituting 1.7 million lines of PL/SQL code. The batch jobs are continuously changing during maintenance and in general they are very complex, hard to test, and prone to regression faults. Since the system serves all taxpayers in Norway, it is vital for the Norwegian Tax Department to avoid releasing defects in the core of the system.

In [5], we proposed an approach and tool (DART) for the regression testing of large database applications like SOFIE, with an emphasis on proving efficient mechanisms for testing oracles, which is the comparison of test results across releases. We presented an evaluation of DART in eight consecutive releases of SOFIE, showing promising fault detection capabilities. In this paper, we will build on this work and devise more sophisticated regression test selection strategies based on black-box models of the input domain.

In a system like SOFIE, vast amounts of data are available from the production environment. Consequently, the testing of SOFIE relies heavily on the use of real input data, leading to a test strategy based on usage profiles. Generating and maintaining such large amounts of synthetic test data would be a tedious and expensive process and it is therefore beneficial to rely on actual tax payer data for regression testing as well. In practice the test data is made available by making a copy of the production database for testing purposes and then reusing input files from the production environment. Thus, the set of test data will vary over time depending on the operations in the production environment. Since we do not rely on exactly the same set of test data over time, we need a cost-effective and

automated strategy for selecting a subset of it that will maximize the likelihood of detecting faults. Several motivating factors exist for doing so. For one thing, the test environments are less sophisticated than the production environment regarding computer resources, thus leading to limitations in terms of the number of test cases that can be run during regression testing. Second, the set of production test data available is likely to contain large numbers of similar test cases, which are thus likely to reveal the same faults. Third, DART produces a set of deviations as regression testing outputs. Testers need to inspect them in order to identify potential regression faults as it is impossible to automatically decide whether such a deviation should be expected given changes made in the last system version. Since such inspections are costly, we would like to reduce the number of deviations to inspect in the regression test. By selecting test cases in an effective manner, the number of deviations resulting from the same fault would hopefully be limited.

Our priority in this paper is therefore to devise and evaluate cost-effective strategies to select regression test cases. Our approach to selection is black-box (specification-based), which is based on a model of the input domain. Choosing a black-box approach for a regression testing of the batch jobs in SOFIE was primarily motivated by the fact that the testers in the project have limited technical expertise regarding the system implementation, and prefer to verify system functionality based on the specifications rather than the source code. Hence, we adopted a black-box approach to regression testing that does not require source code analysis [5].

3. Background

The literature within the area of regression test case selection has primarily been focused on white-box techniques, and coverage-based selection has been the most common practice for years [1]. However, these techniques are not directly applicable within the context of black-box testing. Hemmati et al. (2010) [3] conducted a detailed study on similarity-based test case selection for model-based testing. Given a scenario in which you have to select a subset of test cases to execute, similarity-based test case selection aims to select the most diverse ones in order to increase the fault detection rate. Hemmati's approach includes three components: an encoding (representation) of test cases, a similarity function, and a selection algorithm. The study showed very good results for using diversity as a means of selecting test cases. Hemmati also investigated different encoding strategies, similarity functions and selection algorithms in two industrial settings and in the context of model-based testing. Overall, results suggested that the Gower Legendre similarity function and the (1+1) Evolutionary selection algorithm was best suited for a similarity-based test case selection on test cases generated from UML state machine models [6, 7]. Cartaxo et al (2011) [4] also reported on the success of using similarity functions as a means of selecting test cases for labeled transition systems.

3.1. Similarity measures

Whereas Hemmati et al. focus on selecting a subset of test cases generated from UML state machines, we will select test cases from classification tree models for black box regression testing of database applications. Hence, it is necessary to introduce similarity measures more closely adapted to our context. As further elaborated in Section 4, the abstract test cases generated from the classification tree models are represented as series of model property types as integers, strings or Boolean values. Given the nature of these test cases, the family of geometrical similarity functions seems to be the most reasonable choice for defining similarity, since each model property could span out a dimension in the N-dimensional space. There are various such functions available, but the ones we selected for our study are *Euclidian distance*, *Manhattan distance*, *Mahalanobis distance* and *Normalized Compression Distance (NCD)*. The rationale behind this selection was to assess functions of varying complexity and assess the trade-off between complexity and selection effectiveness. Euclidian and Manhattan are simple distance measures, while Mahalanobis is a bit more sophisticated, taking the correlation between model properties into account. Despite being a different domain of application, Liparas et al. (2011) [8] recently reported good results when using Mahalanobis for defect diagnosis. NCD uses a general compression format (e.g. gzip) to group various types of objects based on their similarity. Since NCD bases its similarity measurements on a general compression format, the method is general and can be used to compare any two objects, regardless of their nature. For example, NCD has been successfully applied to cluster music files based on genres and even by composer within the specific genre of classical music [9]. A more formal presentation of each similarity measure is given below.

The Euclidian distance between two points $\vec{x}$ and $\vec{y}$ is the length of the line segment connecting them [10], given by the following formula:

$$\text{Euc}(\vec{x}, \vec{y}) = \sum_{i=1}^{n} \sqrt{(y_i - x_i)^2} \tag{1}$$

The Manhattan distance between two points $\vec{x}$ and $\vec{y}$ is the sum of the absolute differences of their coordinates [11], given by the following formula:

$$\text{Man}(\vec{x}, \vec{y}) = \sum_{i=1}^{n} |x_i - y_i| \tag{2}$$

The Mahalanobis distance between two points $\vec{x}$ and $\vec{y}$ is given by the following formula [12]:

$$\text{Mah}(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T S^{-1} (\vec{x} - \vec{y})} \tag{3}$$

where S is the covariance matrix. A simplistic approach to estimating the probability that a test point in an N-dimensional space belongs to a set, is to average the center of the sample points and use the standard deviation to determine the probability. While this approach assumes a spherical distribution, Mahalanobis accounts for non-spherical distribution (i.e. ellipsoidal) by incorporating the covariance matrix. Thus it does not only take the distance from the center into account, but also the direction.

Bennett et al. have introduced a universal cognitive similarity distance called Information Distance [13]. Information Distance, which is based on Kolmogorov complexity, is, however, uncomputable. The uncomputability of the Information Distance metric can be overcome by using data compressors to approximate Kolmogorov complexity. i.e. compressors like gzip and bzip2. In [14], Cilibrasi introduced the Normalized Compression Distance, NCD, which uses compressors to approximate Kolmogorov complexity and, thus, also provides practitioners a way to indirectly use the Information Distance metric:

$$\text{NCD}(x, y) = \frac{C(x, y) - min\{C(x), C(y)\}}{max\{C(x), C(y)\}} \tag{4}$$

where $C(x)$ is the length of the binary string x after compression by the compressor C and $C(xy)$ is the length of the concatenated binary string xy after compression by the compressor C. In practice, NCD is a non-negative number $0 <= r <= 1 + \epsilon$, where ϵ is small and depends on how good an approximation of Kolmogorov complexity the compressor (C) is. For modern compressors like gzip and bzip2, ϵ is typically 0.1 and more recent compressors can even come close to 0, i.e. being excellent approximations of Kolmogorov complexity.

3.2. Selection algorithms

The best selection algorithm, as reported in the study of Hemmati et al. was the *(1+1) evolutionary algorithm (EA)* [3]. Evolutionary algorithms imitate evolution as it is observed in nature, where the repeated process of recombination, mutation, and selection leads to individuals that are increasingly adapted to their environment [15]. In evolutionary algorithms, possible solutions to the optimization task are called individuals, and a set of individuals is called a population. For example, in our context, an individual is a set of test cases. (1+1) EA is a simple variant of evolutionary algorithms that restricts the population size to just one individual. Thus, the evolution starts from a single individual, generally chosen at random, which evolves through generations. At each generation (iteration) a single offspring is generated by mutating the parent, but the offspring will not replace their parents if they have a worse fitness value. In practice this means that each test case is mutated with a probability of $1/n$, where n is the number of test cases in the individual. A mutated test case is replaced by a randomly chosen test case among the ones that are not already included. After each mutation process, the fitness value of the new population is evaluated. The fitness value is the sum of all pairwise similarities between test cases, given by the similarity function. The new generation only replace the current population if the fitness value indicates a higher degree of diversity. The evolution stops when a stop criterion is met, expressed by either a time constraint or a maximum number of iterations. Throughout the paper, we will refer to (1+1) evolutionary algorithm as the *evolutionary selection algorithm*.

Since our area of application is similar to that of Hemmati et al. (i.e., test case selection), we will also rely on *(1+1) EA*. We also chose to include in our study a simpler, more commonly used algorithm, namely *greedy-based*

minimization, using it as a baseline of comparison to assess whether an evolutionary algorithm is indeed needed given its additional complexity. When selecting n test cases out of a test suite of N test cases, greedy-based minimization will remove the $N - n$ test cases with the largest pairwise similarity value. It will do so at each step of the algorithm by selecting the two test cases that have the largest similarity value. If more than one pair of test cases evaluates to the same maximum similarity value, a random one is chosen. The greedy approach continuously selects the two most similar test cases, but does not necessarily maximize the combined similarity of the $N - n$ test cases removed. Throughout the paper, we will refer to greedy-based minimization as the *greedy selection algorithm*.

4. Proposed solution

In order to define the test specifications for the system being tested, we relied on classification tree models, which is a well-known black-box specification technique [2]. The Norwegian tax department chose to use CTE-XL for their classification tree modeling. CTE-XL is a commercial tool partly built on the ideas of the category-partition method by Ostrand and Balcer [16], and is used to partition a test domain and generate a test specification (consisting of a set of abstract test cases). In CTE-XL, we model the input domain of the system being tested as a classification tree (step 1), where all relevant distinguishing properties are captured at the desired granularity level. More specifically, properties related to the input domain that may affect the behavior of SOFIE are identified, and equivalence classes are defined for each property following the usual black-box testing strategies, such as boundary value analysis for example. Next, we generate abstract test case specifications from the model (step 2), which are valid combinations of equivalence classes in the model. Once the model is established and the abstract test cases are generated, we locate the test data available (production data in our industry context) in the system for a particular test, and match this data with the abstract test cases (partitions) from the model specification (step 3).

Let us examine an illustrative example. The upper right corner of Figure 1 shows a classification tree model for the functional domain A. It is a simple model containing three properties, (1) property A (number of X), with two equivalence classes: 1–4 and 5–10, (2) property B (has Z), with two equivalence classes: Yes and No, and (3) property C (number of Y), with three equivalence classes: 0, 1 and > 1. The lower part of Figure 1 shows the abstract test cases (partitions), which are all valid combinations of the equivalence classes in the model. The model and the corresponding partitions are the outcome of Steps 1 and 2, as described in the previous section. Table 1 contains an example set of available test data from a particular test. When matching the available test data with the partitions in the model, test case 1–10 will be contained in partition 3, as the value of property A is within the range of 1–4, the value of property B is Yes and the value of property C is > 1. Similarly, test case 11–20 will be contained in partition 12. With the domain model, the abstract test cases, and the set of available test data at hand, the next step is to select the test cases to execute in the regression test.

4.1. Random partition-based test case selection

In our previous study [5], we applied the following simple strategy for selecting test cases for regression testing, called random partition-based test case selection: Among the partitions containing test cases, we selected a random partition, and then a random test case within the partition. Next, we again selected a random partition among the remaining ones, and again a random test case within the partition. This process continued until all the partitions were selected. Then the process was then started again and test cases selected from among the ones that had not been selected yet. The selection process stopped when the desired number of test cases were selected.

Despite being more efficient than random selection, this test case selection strategy is likely to be sub-optimal, since it selects test cases at random within each partition. Referring to the example test cases in Table 1, a random partition-based selection strategy would have the same likelihood of selecting test cases 1 and 2 from partition 3 as test cases 1 and 10, although intuitively it seems more likely that test case 1 and 10 will reveal different faults than test cases 1 and 2, because their property values are more diverse. Therefore, we are curious to see whether the notion of similarity measurements could improve the test case selection process, and hypothesize an improvement in the fault detection rate.

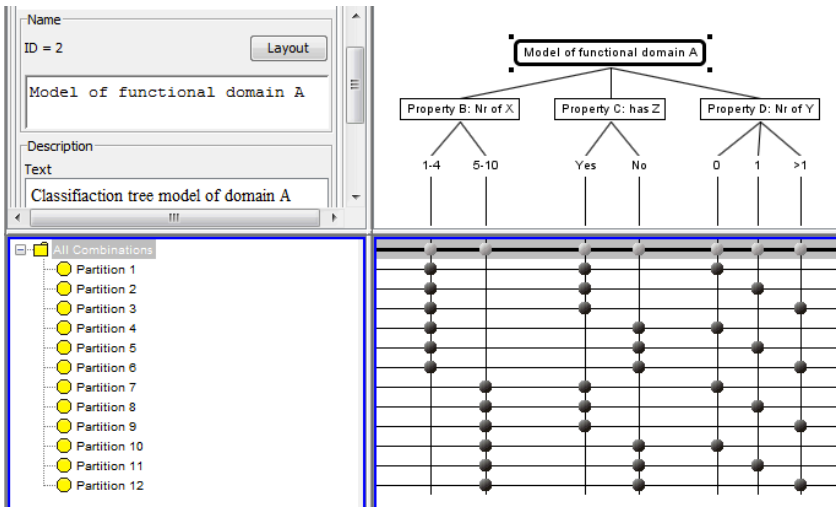


Figure 1: Example of a classification tree model.

Table 1: Example test cases from partition 3 in Figure 1.

Testcase	Property A: Nr of X	Property B: Has Z	Property C: Nr of Y
Testcase 1	1	Yes	2
Testcase 2	1	Yes	2
Testcase 3	1	Yes	3
Testcase 4	2	Yes	2
Testcase 5	2	Yes	2
Testcase 6	2	Yes	4
Testcase 7	4	Yes	2
Testcase 8	4	Yes	2
Testcase 9	4	Yes	3
Testcase 10	4	Yes	4
Testcase 11	5	No	2
Testcase 12	5	No	2
Testcase 13	5	No	3
Testcase 14	7	No	2
Testcase 15	7	No	2
Testcase 16	7	No	3
Testcase 17	7	No	4
Testcase 18	8	No	2
Testcase 19	9	No	2
Testcase 20	10	No	3

Table 2: Test case encoding example.

(a) Example of value-based encoding of test cases.

Testcase	Property A: Nr of X	Property B: Has Z	Property C: Nr of Y
Testcase 1	1	Yes	2

(b) Example of binary encoding of test cases.

Testcase	A: 1-4	A: 5-10	B: Yes	B: No	C: 0	C: 1	C: $\gg 1$
Testcase 1	1	0	1	0	0	0	1

4.2. Similarity measurement for classification tree models

In order to apply similarity based test case selection, the test cases need to be encoded (1), a similarity function is needed to quantify the similarity between test cases (2), and a selection algorithm is needed to select the most diverse test cases (3).

There are two ways to encode a test case based on a classification tree model. The test case could either be encoded with its actual value for a given property, or it can be encoded as a Boolean matrix where for each equivalence class in each property, a true or false value is assigned. For example test case 1 from the example in Section 4 could either be encoded as having the values (1, *Yes*, 2) for property *A*, *B*, and *C*, respectively, or it could be encoded as (T, F, T, F, F, F, T), assigning true (T) or false (F) for all the equivalence classes in the model. The two variants of encoding are shown in Table 2. In order to introduce similarity-based selection within each partition, the only meaningful way to go is to use an encoding based on actual values. The Boolean encoding does not capture enough information to differentiate the various test cases within the same partition, as it is just a direct reflection of the model.

Given the value-based encoded test cases, a method for defining similarities between test cases is needed. Regardless of the similarity function, we have to make some choices about how to apply it for the purposes of selection. The subject model in this study does not contain any string values, but is limited to integers and boolean values. Thus, we can apply the similarity functions directly. (Further elaboration of different strategies for string matching can be found in [17].) If the property values in the model vary to a great extent, they should be normalized to prevent the small values from being obfuscated by much larger values. The output of the similarity measurement is a similarity matrix expressing the pair-wise similarity between all test cases. Having decided on how to encode the test cases and how to apply the similarity functions, we investigated two approaches of similarity-based test case selection, which will now be presented.

4.3. Pure similarity-based test case selection

One alternative is to solely rely on similarity-based test case selection strategy. By that, we mean selecting test cases from the test suite without accounting for the partitions given from the domain model. We take the entire test suite as an input, encode the test cases, define pair-wise similarity between all the test cases in a similarity matrix, and then focus on selecting the most diverse ones. That is: given a test suite of N test cases, generate an $N \times N$ symmetric similarity matrix and select the n most diverse test cases. The set of 'most diverse' test cases will vary depending on the selection algorithm, whereas the values of the similarity matrix will vary with the similarity function. When using a pure similarity-based test case selection strategy it is not necessary to carry out steps 2 and 3 described in section 4, namely to generate the partitions and match the test data with the partitions. So rather than selecting test cases from partitions defined using human expertise, the test cases are selected purely based on similarity measures (diversity).

4.4. Similarity partition-based test case selection

Another alternative is to combine similarity measurements with partition-based selection, denoted as similarity partition-based test case selection. Rather than selecting test cases within each partition in a completely random manner, we could incorporate a similarity-based test case selection strategy within each partition. Incorporating similarity measurements within the partitions would ensure that we select as diverse test cases as possible from each partition, thus benefiting from both the model partitions and the notion of diversity within each partition. A similarity partition-based strategy generates one similarity matrix per partition containing test cases, rather than generating one for the whole test suite, and uses it to select the most diverse test cases within each partition. Applying the concept of similarity measures within each partition therefore has the advantage of keeping the similarity matrices smaller and less resource intensive.

To exemplify, when selecting four test cases from the available example test cases in Table 1, *random partition-based* would select two random test cases from each partition, *pure similarity-based* would select the four most diverse

test cases of the entire test suite, while *similarity partition-based* would select the two most diverse test cases from each partition.

5. Experiment

This section presents an experiment aiming at investigating the use of similarity-based test case selection for the black-box regression testing of database applications. We will elaborate on the design and analysis of the experiment, report the results, and discuss the outcome.

5.1. Research questions

Our goal is to determine which similarity-based selection algorithm fares best and then assess whether and under which conditions similarity-based selection is a worthwhile and practical alternative to simpler solutions. The latter include random selection as this is by far the easiest and least expensive selection technique to apply and any alternative must be justified by significant improvements. The research questions for this study are as follows:

RQ1 For each selection algorithm, which similarity function, i.e. Euclidian, Manhattan, Mahalanobis and NCD, is best suited for defining similarity between test cases, in order to obtain the best fault detection rate and selection execution time when performing similarity-based selection of test cases generated from classification tree models?

RQ2 When used in combination with their best similarity function (RQ1), which one of the selection algorithms, i.e., evolutionary and greedy, provides better fault detection rates and selection execution times, when selecting test cases generated from classification tree models?

RQ3 Which one of the selection strategies, random partition-based, similarity partition-based and pure similarity-based (the two latter ones incorporating the best combination of similarity function —RQ1—and selection algorithm—RQ2) provides better fault detection rates and selection execution times when selecting test cases generated from classification tree models?

RQ4 Comparing the best selection approach from RQ3 and a random approach, which one provides better fault detection rates and selection execution times when selecting test cases generated from classification tree models?

5.2. Design and analysis

The subject test suite for the experiment contains 5,670 test cases, split across 130 partitions in the particular model for the domain under test. The test suite is based on actual data from the production environment in the SOFIE project, and was the only available input file for the particular regression test under consideration. For evaluation purposes we ran the regression test on the entire test suite. The regression test compares two runs of a baseline and delta version of the system under test, resulting in a set of deviations that indicate a change (regression fault or a correct change). More information on the specifics of the regression test is available in [5]. The subject test suite in this study resulted in 522 deviations and of these, 136 deviations were faulty, capturing 15 distinct faults. All the faults are real faults (no seeded faults), emerging from a test suite in industry.

In order to determine the best similarity function and selection algorithm, we ran all combinations of similarity functions and selection algorithms for the entire test suite. We applied the similarity functions Euclidian, Manhattan, Mahalanobis and NCD to generate a similarity matrix for the entire test suite, and in turn used each of them as inputs for both the evolutionary and greedy selection algorithms. Each combination of similarity function and selection algorithm was used to select subsets of test cases, ranging from 5% to 95% of the entire test suite, with 5% intervals. For each subset we logged the exact test cases selected, and reported the number of distinct faults revealed, along with the selection execution time. We repeated the exercise 175 times in order to gain satisfactory statistical power when comparing similarity functions and selection algorithms for RQ1 and RQ2. Specifically, we scheduled the experiment to run in parallel on a cluster and the maximum number of nodes available was 175. With one parallel run on each node, this led to 175 repetitions, which was deemed sufficient to detect differences large enough to be of practical interest.

Once the outcome of RQ1 and RQ2 was concluded, the pure similarity-based approach was given directly, and constituted of the best combination of similarity function and selection algorithm. The similarity partition-based approach was composed by incorporating the best combination of similarity function and selection algorithm with the partition-based approach. Again we selected a subset of test cases ranging from 5% to 95% of the entire test suite, with 5% intervals, logged the same data points and repeated the exercise 175 times for random partition-based, similarity partition-based, pure similarity-based and random in order to address RQ3 and RQ4.

As suggested in [18] and [19], we selected a non-parametric statistical test since we could not fulfill the underlying assumption of normality and equal variance between our data samples. More specifically, we used a two-tailed Mann-Whitney U-tests (Wilcox test in R) to conduct pair-wise algorithm comparisons for all sample values, ranging from 5% to 95%. The p-values for each comparison is reported, and $\alpha = 0.05$ is used whenever referring to statistical significance. However, a Mann-Whitney U-test reports only whether there is a statistically significant difference between two algorithms, but does not clarify the magnitude of the difference. Thus, we use the $\hat{A}_{12}$ effect size measure to assess the practical significance of differences. An $\hat{A}_{12}$ effect size measurement value of 0.5 indicates that there is no difference between the two samples compared, whereas values above 0.5 indicates that sample A is superior to sample B, and opposite for values smaller than 0.5. The further away from 0.5, the larger the effect size. The value of the effect size measurement is reported, but for increased visibility we have also categorized the effect size into Small, Medium and Large. The categories resolves to the following interpretation: Small < 0.10, 0.10 < Medium < 0.17 and Large > 0.17, the value being the distance from the 0.5 value [19].

5.3. Results

This section will report the results from testing the experimental hypotheses that can be derived from the research questions in Section 5.1.

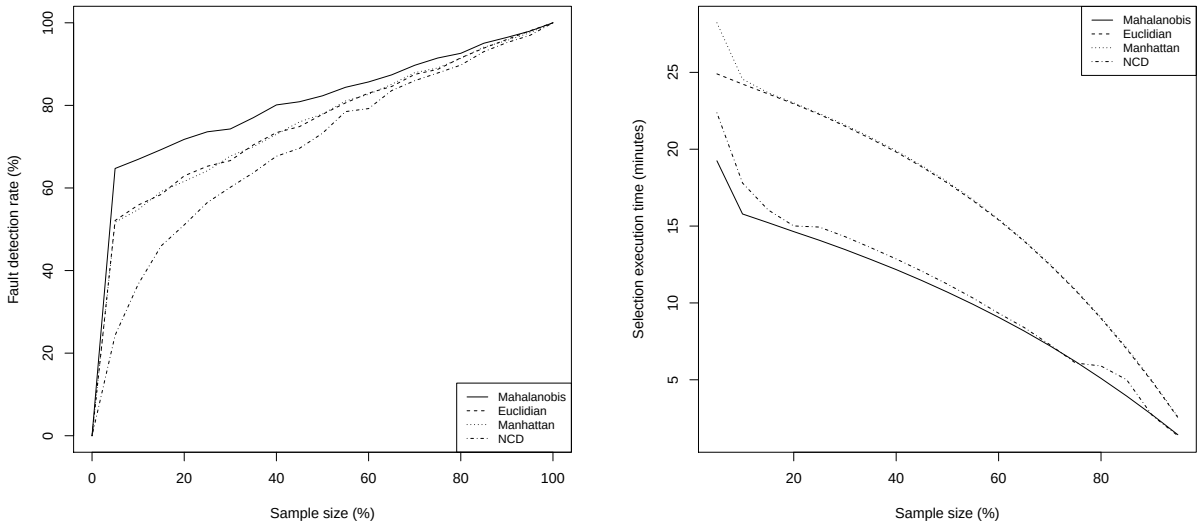
Throughout this section the results will be reported as follows:

- A graph showing the average percentage of faults found (y-axis) per percentage of selected test cases (x-axis).
- A graph showing the average selection execution time (y-axis) in minutes per percentage of selected test cases (x-axis).
- A table showing the results of the statistical tests and effect size measurement for each algorithm comparison. The table will adhere to the following structure: Column one indicates the sample size in percentage. Then each algorithm comparison will be represented in one column, which is split into three sub-columns. The first sub-column reports the p-value from the Mann-Whitney U-test, the second sub-column reports the *superior* algorithm of the two, and the third column reports the $\hat{A}_{12}$ effect size measurement, both the actual value and on a three-point scale for helping the visualization of trends, i.e. Small, Medium or Large. The same table structure will be used to report comparisons of both fault detection rate and selection execution time.

5.3.1. Research question 1—Similarity functions

The results for each of the similarity functions combined with the greedy selection algorithm are depicted in Figure 2, Table 3. By looking at the graphs in Figure 2(a) and Figure 2(b), it seems evident that Mahalanobis is the best similarity function for the greedy selection algorithm, since it has the highest fault detection rate and the lowest selection execution time. This observation is confirmed by the conducted statistical tests and the measured effect size (only the results regarding fault detection rate are reported), shown in Table 3. In terms of fault detection rate reported in Table 3, Mahalanobis is significantly higher than Euclidian and Manhattan for all sample sizes up to 70% and higher, than NCD for all sample sizes. The effect size is large for smaller sample sizes, medium for medium-sized sample sizes and small for larger sample sizes, all in favor of Mahalanobis, except for one sample value at 95%. The results also indicate that there is no significant difference in the fault detection rate between the Euclidian and Manhattan similarity functions, and there is no consistency in the effect size reported. NCD is worse than the three others for all sample sizes.

In terms of selection execution time, Mahalanobis and NCD is significantly quicker than both Euclidian and Manhattan for all sample sizes, and the effect size is large. Mahalanobis is quicker than NCD for the lower sample sizes, whereas NCD is faster for the large sample sizes. Between Euclidian and Manhattan, Euclidian shows a better selection execution time, but with varying significance and effect sizes. A general note regarding the selection execution



(a) Average number of faults found (y-axis) per number of test cases selected (x-axis) for the greedy selection algorithm.

(b) Average selection execution time (y-axis) per number of test cases selected (x-axis) for the greedy selection algorithm.

Figure 2: Graphs comparing different similarity functions for the greedy selection algorithm.

time is that the greedy selection algorithm performs increasingly better the closer it gets to 100% of the sample size. This is because the greedy algorithm excludes the most similar test cases rather than selecting the most diverse ones, i.e. when selecting n test cases from a sample of N test cases, the greedy algorithm would exclude $N - n$ test cases rather than select n test cases.

Given the results discussed above, it is important to use the Mahalanobis similarity function in combination with the greedy selection algorithm in order to obtain the best fault detection rates and selection execution time.

Figure 3 and Table 4 depict and report the results for each of the similarity functions combined with the evolutionary selection algorithm. As the graph in Figure 3(b) show, there is no practical difference between the similarity functions regarding selection execution time (the lines are virtually on top of each other). The same accounts for the fault detection rate between Euclidian, Manhattan and Mahalanobis, while NCD falls short of the others, as shown in Figure 3(a). The statistical tests (only the results regarding fault detection rate are reported) confirm that Euclidian, Manhattan and Mahalanobis are better than NCD for all sample sizes and the effect size is large. Between the former three, the null-hypothesis stating that there is no difference in fault detection rates between the algorithms cannot be rejected other than for the sample sizes 5% to 15% in the Mahalanobis vs. Euclidian comparison and for the sample sizes 5%, 10%, 25%, 35%, 60%, 75% and 80% in the Mahalanobis vs. Manhattan comparison, all in favor of Mahalanobis. Regarding selection execution time, there is no notable difference among algorithms. The results for the evolutionary selection algorithm do not show clear differences as for the greedy approach, except for the poor results of NCD, but if we have to choose one, Mahalanobis is again the preferred similarity function. Despite not being superior for all sample sizes, Mahalanobis is significantly better for smaller sample sizes, which are the most important ones, and it is not inferior to the others for any other sample sizes.

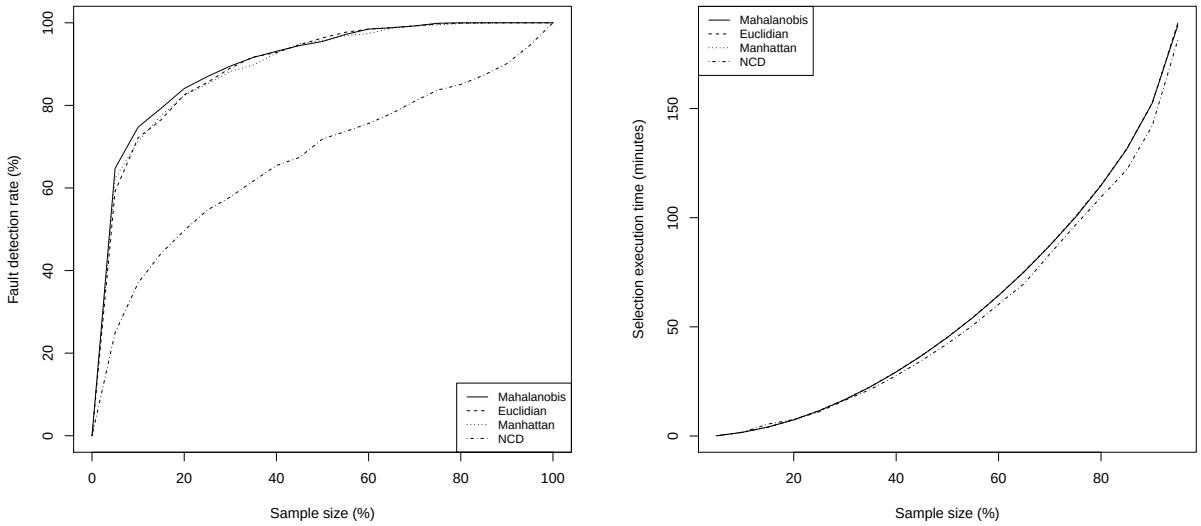
In summary, we can address RQ1 by stating that Mahalanobis should be the preferred similarity function for both the greedy and evolutionary selection algorithms. This could indicate that it is important to consider correlation among model properties, since incorporation of correlation is the main differentiating factor between the Mahalanobis similarity function and the two simpler similarity functions Euclidian and Manhattan. NCD falls short of the others in terms of the fault detection rate. The most plausible reason is that the test cases are represented by a fairly simple structure that accounts for very little information distance between their compressed versions. Thus, NCD is not able to pick up the minor differences as well as simpler geometrical functions.

Table 3: Mann-Whitney U-tests and $\hat{A}_{12}$ effect size measurements when comparing fault detection rate across the similarity functions Euclidian, Manhattan, Mahalanobis and NCD for the greedy selection algorithm.

Algorithm comparison / Sample size (%)	Mahalanobis (A), Euclidian (B)			Mahalanobis (A), Manhattan (C)			Euclidian (B), Manhattan (C)		
	p-value	**Superior**	**Effect size**	**p-value**	**Superior**	**Effect size**	**p-value**	**Superior**	**Effect size**
5	2.1728e-44	A	Large (0.9207)	4.3911e-48	A	Large (0.9392)	0.7158	B	Small (0.4894)
10	1.3490e-36	A	Large (0.8803)	7.5116e-41	A	Large (0.9037)	0.1466	B	Small (0.4572)
15	1.4171e-31	A	Large (0.8528)	6.1495e-30	A	Large (0.8423)	0.4149	C	Small (0.5241)
20	4.0909e-21	A	Large (0.7840)	1.1413e-27	A	Large (0.8284)	0.1388	B	Small (0.4561)
25	9.2537e-19	A	Large (0.7645)	1.9528e-22	A	Large (0.7931)	0.1459	B	Small (0.4568)
30	1.8968e-17	A	Large (0.7557)	2.0203e-12	A	Large (0.7117)	0.1953	C	Small (0.5389)
35	7.1533e-11	A	Large (0.6969)	2.7877e-12	A	Large (0.7109)	0.7047	B	Small (0.4886)
40	1.0095e-12	A	Large (0.7142)	4.9575e-14	A	Large (0.7271)	0.4939	B	Small (0.4795)
45	5.4848e-10	A	Large (0.6871)	2.6814e-07	A	Medium (0.6551)	0.2968	C	Small (0.5315)
50	1.6722e-06	A	Medium (0.6444)	8.5201e-06	A	Medium (0.6343)	0.8441	C	Small (0.5059)
55	3.5057e-05	A	Medium (0.6243)	0.0003	A	Medium (0.6088)	0.5831	C	Small (0.5165)
60	0.0033	A	Small (0.5879)	0.0001	A	Medium (0.6137)	0.6210	B	Small (0.4852)
65	0.0011	A	Small (0.5974)	0.0147	A	Small (0.5732)	0.4950	C	Small (0.5205)
70	0.0053	A	Small (0.5829)	0.0459	A	Small (0.5593)	0.5272	C	Small (0.5190)
75	0.0020	A	Small (0.5920)	0.0030	A	Small (0.5878)	0.7378	C	Small (0.5099)
80	0.1215	A	Small (0.5457)	0.1228	A	Small (0.5456)	0.9727	B	Small (0.4990)
85	0.0356	A	Small (0.5602)	0.1770	A	Small (0.5385)	0.4723	C	Small (0.5207)
90	0.3730	A	Small (0.5244)	0.0550	A	Small (0.5532)	0.3218	B	Small (0.4723)
95	0.5088	B	Small (0.4845)	0.3755	A	Small (0.5218)	0.1308	B	Small (0.4636)
	Mahalanobis (A), NCD (D)			**Euclidian (B), NCD (D)**			**Manhattan (C), NCD (D)**		
	p-value	**Superior**	**Effect size**	**p-value**	**Superior**	**Effect size**	**p-value**	**Superior**	**Effect size**
5	8.9768e-57	A	Large (1)	5.8001e-56	B	Large (0.9970)	4.9039e-56	C	Large (0.9967)
10	2.7331e-56	A	Large (0.9991)	1.4736e-46	B	Large (0.9515)	3.6345e-45	C	Large (0.9440)
15	8.6973e-54	A	Large (0.9879)	2.9097e-29	B	Large (0.8509)	1.3015e-31	C	Large (0.8655)
20	2.5482e-46	A	Large (0.9509)	7.8675e-25	B	Large (0.8227)	3.5295e-21	C	Large (0.7952)
25	1.2666e-37	A	Large (0.9038)	1.9739e-17	B	Large (0.7647)	2.6640e-13	C	Large (0.7282)
30	3.1773e-34	A	Large (0.8831)	1.2455e-10	B	Large (0.7001)	1.6870e-13	C	Large (0.7297)
35	7.0757e-30	A	Large (0.8564)	7.4680e-10	B	Large (0.6922)	2.8742e-10	C	Large (0.6967)
40	3.3052e-26	A	Large (0.8318)	1.1187e-07	B	Medium (0.6651)	6.6678e-06	C	Medium (0.6402)
45	3.1297e-23	A	Large (0.8114)	7.9522e-07	B	Medium (0.6543)	5.7825e-09	C	Large (0.6821)
50	4.5072e-15	A	Large (0.7460)	1.7755e-05	B	Medium (0.6342)	1.0674e-05	C	Medium (0.6379)
55	1.6341e-09	A	Large (0.6879)	0.0366	B	Small (0.5650)	0.0145	C	Small (0.5760)
60	4.2518e-13	A	Large (0.7245)	0.0003	B	Medium (0.6116)	0.0002	C	Medium (0.6125)
65	2.4608e-05	A	Medium (0.6306)	0.2085	B	Small (0.5390)	0.0912	C	Small (0.5524)
70	6.1526e-06	A	Medium (0.6389)	0.1165	B	Small (0.5483)	0.0275	C	Small (0.5682)
75	1.0828e-05	A	Medium (0.6355)	0.2853	B	Small (0.5330)	0.1071	C	Small (0.5494)
80	0.0014	A	Small (0.5978)	0.0357	B	Small (0.5644)	0.0621	C	Small (0.5573)
85	0.0058	A	Small (0.5823)	0.4329	B	Small (0.5235)	0.0813	C	Small (0.5521)
90	0.0101	A	Small (0.5740)	0.1434	B	Small (0.5425)	0.5825	C	Small (0.5160)
95	0.0357	A	Small (0.5547)	0.0040	B	Small (0.5734)	0.282	C	Small (0.5286)

Table 4: Mann-Whitney U-tests and $\hat{A}_{12}$ effect size measurements when comparing fault detection rates across the similarity functions Euclidian, Manhattan, Mahalanobis and NCD for the evolutionary selection algorithm.

Algorithm comparison / Sample size (%)	Mahalanobis (A), Euclidian (B)			Mahalanobis (A), Manhattan (C)			Euclidian (B), Manhattan (C)		
	p-value	**Superior**	**Effect size**	**p-value**	**Superior**	**Effect size**	**p-value**	**Superior**	**Effect size**
5	2.2100e-09	A	Large (0.6797)	0.01339	A	Small (0.5740)	0.0001	C	Medium (0.6158)
10	0.0033	A	Small (0.5885)	0.0003	A	Medium (0.6103)	0.3552	B	Small (0.4721)
15	0.0042	A	Small (0.5863)	0.0515	A	Small (0.5586)	0.3309	C	Small (0.5293)
20	0.2229	A	(Small (0.5366)	0.0526	A	Small (0.5579)	0.5269	B	Small (0.4810)
25	0.1251	A	Small (0.5459)	0.0253	A	Small (0.5668)	0.5279	B	Small (0.4811)
30	0.5644	A	Small (0.5171)	0.1176	A	Small (0.5467)	0.2847	B	Small (0.4682)
35	0.9075	B	Small (0.4966)	0.0349	A	Small (0.5626)	0.0184	B	Small (0.4307)
40	0.6634	A	Small (0.5126)	0.4593	A	Small (0.5215)	0.7691	B	Small (0.4914)
45	0.7801	B	Small (0.4920)	0.4388	C	Small (0.4778)	0.6201	C	Small (0.5142)
50	0.1646	B	Small (0.4612)	0.6034	C	Small (0.4854)	0.3979	B	Small (0.4766)
55	0.2499	B	Small (0.4705)	0.8279	A	Small (0.5057)	0.1829	B	Small (0.4658)
60	1	None	NO effect (0.5)	0.0039	A	Small (0.5689)	0.0039	B	Small (0.4311)
65	0.7087	A	Small (0.5075)	0.5046	A	Small (0.5136)	0.7657	B	Small (0.4938)
70	0.8927	A	Small (0.5022)	0.7486	A	Small (0.5054)	0.8516	B	Small (0.4968)
75	0.2006	A	Small (0.5114)	0.0106	A	Small (0.5286)	0.1681	B	Small (0.4829)
80	0.1579	A	Small (0.5057)	0.0247	A	Small (0.5143)	0.2536	B	Small (0.4914)
85	NaN	None	NO effect (0.5)	0.0830	A	Small (0.5086)	0.0830	B	Small (0.4914)
90	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)
95	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)
	Mahalanobis (A), NCD (D)			**Euclidian (B), NCD (D)**			**Manhattan (C), NCD (D)**		
	p-value	**Superior**	**Effect size**	**p-value**	**Superior**	**Effect size**	**p-value**	**Superior**	**Effect size**
5	1.8326e-59	A	Large (0.9997)	1.9073e-59	B	Large (0.9994)	1.4201e-59	C	Large (0.9998)
10	1.2037e-59	A	Large (0.9994)	1.0507e-59	B	Large (0.9995)	1.6212e-59	C	Large (0.9991)
15	1.5322e-59	A	Large (0.9997)	4.3178e-59	B	Large (0.9979)	2.2875e-59	C	Large (0.9990)
20	2.0147e-59	A	Large (0.9990)	8.3626e-59	B	Large (0.9965)	1.2182e-59	C	Large (0.9995)
25	1.6968e-59	A	Large (0.9983)	7.4124e-59	B	Large (0.9956)	6.2126e-59	C	Large (0.9956)
30	3.9722e-59	A	Large (0.9980)	7.8319e-59	B	Large (0.9964)	1.1977e-58	C	Large (0.9961)
35	1.4701e-58	A	Large (0.9957)	4.7277e-59	B	Large (0.9964)	3.1509e-57	C	Large (0.9899)
40	1.0006e-59	A	Large (0.9986)	8.7756e-60	B	Large (0.9991)	1.0951e-59	C	Large (0.9991)
45	2.5088e-59	A	Large (0.9967)	1.2940e-59	B	Large (0.9978)	1.8517e-59	C	Large (0.9968)
50	7.6166e-58	A	Large (0.9890)	3.0977e-59	B	Large (0.9937)	7.0280e-58	C	Large (0.9887)
55	9.8988e-61	A	Large (0.9977)	1.5907e-61	B	Large (0.9988)	9.3485e-60	C	Large (0.9937)
60	3.1531e-62	A	Large (0.9964)	3.1531e-62	B	Large (0.9964)	5.1556e-60	C	Large (0.9937)
65	7.8723e-62	A	Large (0.9909)	7.8007e-62	B	Large (0.9916)	2.6002e-61	C	Large (0.9906)
70	1.4747e-62	A	Large (0.9898)	4.8292e-63	B	Large (0.9918)	1.9815e-62	C	Large (0.9902)
75	8.7151e-67	A	Large (0.9985)	1.0672e-65	B	Large (0.9965)	3.6269e-64	C	Large (0.9936)
80	4.8507e-65	A	Large (0.9885)	2.4673e-64	B	Large (0.9870)	2.6303e-63	C	Large (0.9846)
85	9.9167e-57	A	Large (0.9454)	9.9167e-57	B	Large (0.9454)	1.5599e-55	C	Large (0.9425)
90	2.6409e-54	A	Large (0.9310)	2.6409e-54	B	Large (0.9310)	2.6409e-54	C	Large (0.9310)
95	1.1022e-36	A	Large (0.8218)	1.1022e-36	B	Large (0.8218)	1.1022e-36	C	Large (0.8218)



(a) Average number of faults found (y-axis) per number of test cases selected (x-axis) for the evolutionary selection algorithm

(b) Average selection execution time (y-axis) per number of test cases selected (x-axis) for the evolutionary selection algorithm

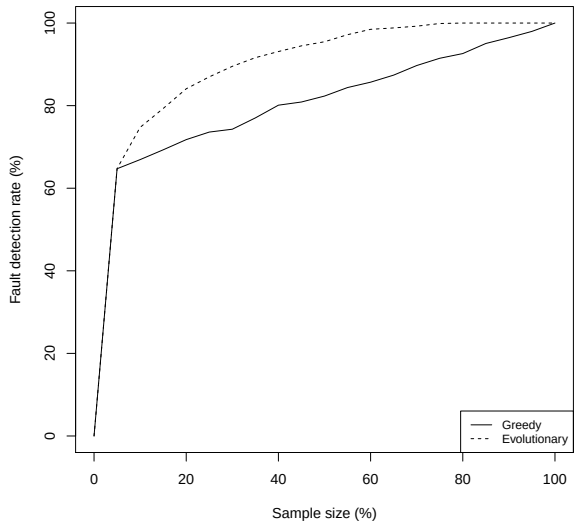
Figure 3: Graphs comparing different similarity functions for the evolutionary selection algorithm.

5.3.2. Research question 2—Selection algorithms

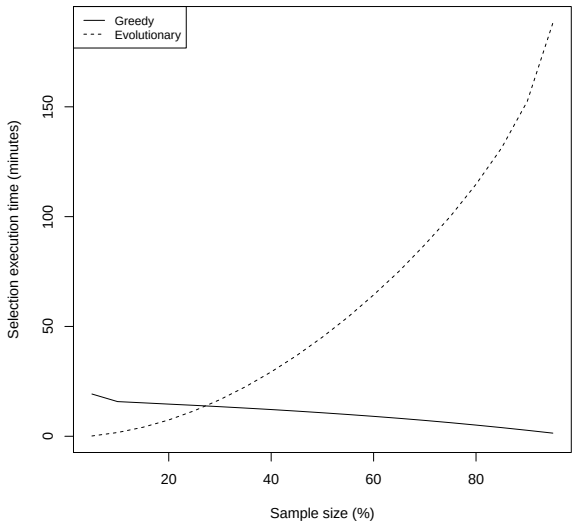
Figure 4 and Table 5 depict and report the results for the greedy and evolutionary selection algorithms, each combined with their best similarity function (Mahalanobis). By looking at the graphs, it seems obvious that evolutionary is better than greedy in terms of fault detection rate, whereas their selection execution time follow different patterns. The selection execution time decreases for the greedy approach as the sample size increases (for the reasons mentioned in Section 5.3.1), whereas it increases for the evolutionary approach. The cross-over point is between sample size 25% and 30%. Regarding fault detection rates, the results reported from the statistical tests and effect size measures show that evolutionary is better than greedy for all sample sizes except 5%, where there is no statistical significance. The effect size is large for all sample sizes except 5% (small) and 95% (medium), still in favor of the evolutionary approach. Also important is the result that evolutionary converges to 100% fault detection much faster than the greedy approach, i.e. it reaches 98.5% and 100% detection rate at 60% and 80% sample sizes, respectively. In terms of selection execution time there is a significant difference for all sample values, in favor of the evolutionary approach for 5% to 25% sample sizes, and in favor of the greedy approach for 30% to 95% sample sizes. Given the superiority of the evolutionary approach in terms of fault detection rate, along with the fact that it performs better for smaller sample sizes, the outcome of RQ2 is that the evolutionary selection algorithm in combination with the Mahalanobis similarity function is the best similarity-based combination for selecting test cases generated from classification tree models in our experiment.

5.3.3. Research question 3—Selection strategies

For RQ3, we are interested in comparing random partition-based with similarity partition-based and pure similarity-based, the latter two incorporating the best combination of similarity function and selection algorithms from RQ2, namely the evolutionary selection algorithm and the Mahalanobis similarity function. Figures 5(a) and 5(b), and Tables 6 and 7 depict and report the results for the similarity partition-based, random partition-based and pure similarity-based approaches. It is clear from the graph that combining similarity measurements with a partition-based approach offers improved selection execution times when compared with a pure similarity-based approach. By using a partition-based strategy, the selection problem is divided into smaller sub-problems. Consequently, the similarity matrices are smaller and easier to work with, e.g. for the 30% sample size, the similarity partition-based approach uses less than



(a) Average number of faults found (y-axis) per number of test cases selected (x-axis) for the best greedy and best evolutionary selection algorithm.



(b) Average selection execution time (y-axis) per number of test cases selected (x-axis) for the best greedy and best evolutionary selection algorithm.

Figure 4: Graphs comparing the best greedy and evolutionary selection algorithm.

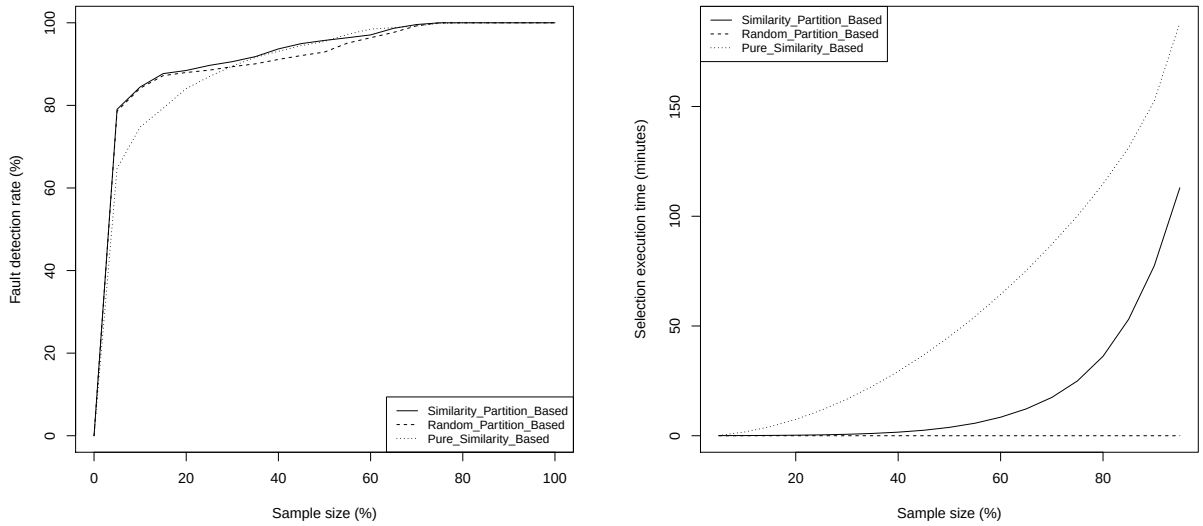
Table 5: Data reported from Mann-Whitney U-tests and $\hat{A}_{12}$ effect size measurements when comparing the greedy and evolutionary selection algorithms.

(a) Mann-Whitney U-tests and $\hat{A}_{12}$ effect size measurements when comparing fault detection rate across the selection algorithms greedy and evolutionary, each combined with their best similarity function.

Comparison / Sample size (%)	Evolutionary (A), Greedy (B)		
	p-value	**Superior**	**Effect size**
5	0.9308	A	Small (0.5026)
10	1.1600e-18	A	Large (0.7650)
15	3.6302e-26	A	Large (0.8191)
20	7.6739e-32	A	Large (0.8561)
25	1.1625e-33	A	Large (0.8677)
30	3.0263e-44	A	Large (0.9240)
35	1.7334e-40	A	Large (0.9049)
40	3.3685e-39	A	Large (0.8962)
45	1.4945e-41	A	Large (0.9085)
50	1.1001e-40	A	Large (0.9027)
55	1.8893e-44	A	Large (0.9180)
60	3.1757e-51	A	Large (0.9431)
65	1.0654e-46	A	Large (0.9164)
70	3.0656e-45	A	Large (0.9007)
75	2.0615e-43	A	Large (0.8720)
80	4.0555e-40	A	Large (0.8457)
85	5.8960e-32	A	Large (0.7886)
90	3.1604e-21	A	Large (0.7057)
95	1.0116e-13	A	Medium (0.6371)

(b) Mann-Whitney U-tests and $\hat{A}_{12}$ effect size measurements when comparing selection execution times across the selection algorithms greedy and evolutionary, each combined with their best similarity function.

Comparison / Sample size (%)	Evolutionary (A), Greedy (B)		
	p-value	**Superior**	**Effect size**
5	1.8315e-63	A	Large (0)
10	5.8220e-59	A	Large (0)
15	6.7460e-59	A	Large (0)
20	6.9419e-59	A	Large (0)
25	2.0272e-58	A	Large (0.0020)
30	2.6063e-58	B	Large (0.9974)
35	6.9358e-59	B	Large (1)
40	7.0241e-59	B	Large (1)
45	7.0198e-59	B	Large (1)
50	7.0340e-59	B	Large (1)
55	7.0580e-59	B	Large (1)
60	7.0387e-59	B	Large (1)
65	7.0340e-59	B	Large (1)
70	6.9589e-59	B	Large (1)
75	7.0392e-59	B	Large (1)
80	6.9336e-59	B	Large (1)
85	6.7849e-59	B	Large (1)
90	6.5583e-59	B	Large (1)
95	5.1709e-59	B	Large (1)



(a) Average number of faults found (y-axis) per number of test cases selected (x-axis) for the three best approaches.

(b) Average selection execution time (y-axis) per number of test cases selected (x-axis) for the three best approaches.

Figure 5: Graphs comparing the three best approaches.

one minute, whereas pure similarity-based uses 22 minutes, which is a substantial improvement in practice. In terms of the fault detection rate, the similarity partition-based curve is steeper than pure similarity-based at the beginning, thus reaching a higher fault detection rate at an earlier stage (for smaller sample sizes). Additionally, it converges even faster to 100% fault detection (at a 75% sample size) and is more reliable since it shows lower variance around the mean (not reported explicitly in the graph). The fault detection rate of the similarity partition-based approach is significantly better than the pure similarity-based approach for smaller sample sizes up to 25%, with a large effect size for 5% to 20%. However, the pure similarity-based approach is significantly better than the similarity partition-based approach for a 60% sample size with a medium effect size. The selection execution time is significantly different for all sample sizes, with a large effect size in favor of the similarity partition-based approach.

The difference between the similarity partition-based and random partition-based is more marginal. The selection execution time is in favor of random partition-based, with significant differences for all sample sizes. However, for sample values up to 50%, execution time ranges from near 0 to up to 4 minutes for similarity partition-based, as opposed to less than a second for random partition-based, a difference which is not of great practical significance. In terms of the fault detection rate, similarity partition-based is significantly better than random partition-based for sample sizes of 15%, between 25% and 55% and for 65%, and is superior for all other sample sizes, though with modest improvements.

To address RQ3, when selecting test cases generated from classification tree models, similarity partition-based provides overall the best fault detection rates, whereas random partition-based shows the lowest selection execution time. It is, however, worth pointing out that the practical significance, which in the end is what decides the usefulness of the approach, is clearly in favor of the prior candidate solution since the selection execution time only comes into play with extremely large test suites.

5.3.4. Research question 4—Compared with a random approach

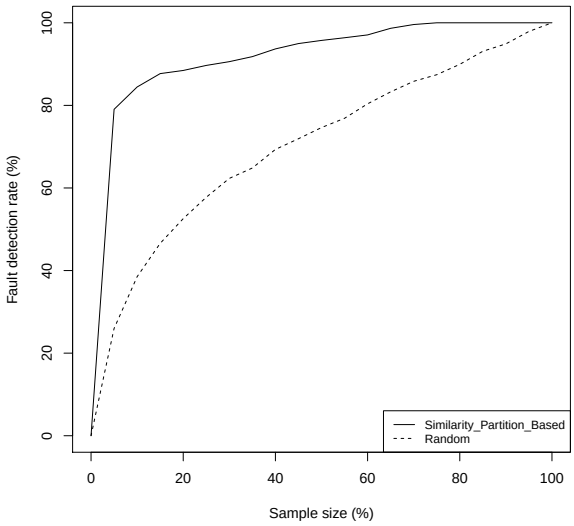
Figure 6 and Table 8 depict and report the results we obtained when comparing similarity partition-based and random test case selection approaches. In terms of fault detection, the improvements when adopting a more sophisticated test case selection strategy such as similarity partition-based are highly significant compared to using a random test selection strategy. As the graph shows, at a sample size of 5%, random selection would on average identify 25% of

Table 6: Mann-Whitney U-tests and $\hat{A}_{12}$ effect size measurements when comparing fault detection rates across the similarity partition-based, random partition-based and pure similarity-based test case selection strategies.

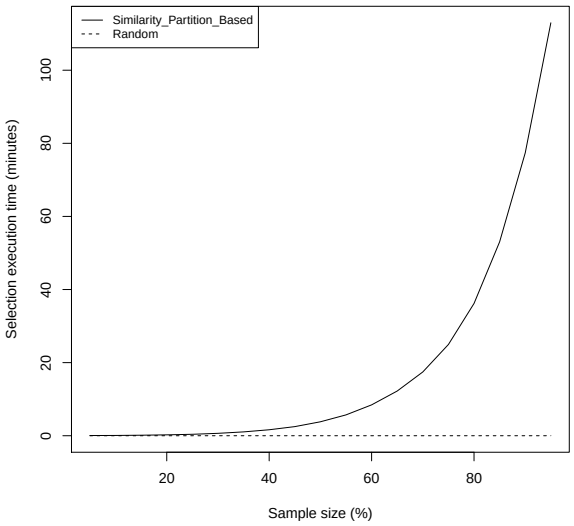
Algorithm comparison / Sample size (%)	Similarity Partition-Based (A), Random Partition-Based (B)			Similarity Partition-Based (A), Pure Similarity-Based (C)			Random Partition-Based (B), Pure Similarity-Based (C)		
	p-value	Superior	Effect size	p-value	Superior	Effect size	p-value	Superior	Effect size
5	0.4677	A	Small (0.5199)	1.7479e-43	A	Large (0.9169)	5.8762e-44	B	Large (0.9159)
10	0.4594	A	Small (0.5199)	3.2483e-29	A	Large (0.8310)	7.2785e-28	B	Large (0.8225)
15	0.0428	A	Small (0.5345)	1.4428e-28	A	Large (0.8110)	1.3581e-27	B	Large (0.7992)
20	0.0984	A	Small (0.5367)	1.1857e-09	A	Large (0.6716)	2.7679e-08	B	Medium (0.6537)
25	0.0023	A	Small (0.5776)	0.0019	A	Small (0.5895)	0.1624	B	Small (0.5395)
30	0.0017	A	Small (0.5852)	0.3066	A	Small (0.5296)	0.3330	C	Small (0.4722)
35	3.7373e-05	A	Medium (0.6141)	0.7949	C	Small (0.4925)	0.0019	C	Small (0.4104)
40	1.6387e-09	A	Medium (0.6624)	0.6126	A	Small (0.5138)	4.2395e-05	C	Medium (0.3827)
45	2.4041e-10	A	Large (0.6715)	0.9072	C	Small (0.4968)	2.6832e-06	C	Medium (0.3641)
50	5.9040e-09	A	Medium (0.6600)	0.5342	C	Small (0.4829)	3.8214e-07	C	Medium (0.3539)
55	0.0109	A	Small (0.5697)	0.0067	C	Small (0.4269)	2.6496e-06	C	Medium (0.3700)
60	0.2574	A	Small (0.5306)	5.6562e-06	C	Medium (0.3863)	1.2307e-07	C	Medium (0.3650)
65	0.0058	A	Small (0.5651)	0.3748	C	Small (0.4817)	0.0004	C	Small (0.4181)
70	0.2287	A	Small (0.5175)	0.1681	A	Small (0.5204)	0.8613	B	Small (0.5028)
75	NaN	None	NO effect (0.5)	0.0830	A	Small (0.5086)	0.0830	B	Small (0.5085)
80	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)
85	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)
90	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)
95	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)	NaN	None	NO effect (0.5)

Table 7: Mann-Whitney U-tests and $\hat{A}_{12}$ effect size measurements when comparing selection execution time across the similarity partition-based, random partition-based and pure similarity-based test case selection strategies.

Algorithm comparison / Sample size (%)	Similarity Partition-Based (A), Random Partition-Based (B)			Similarity Partition-Based (A), Pure Similarity-Based (C)			Random Partition-Based (B), Pure Similarity-Based (C)		
	p-value	Superior	Effect size	p-value	Superior	Effect size	p-value	Superior	Effect size
5	5.6095e-69	B	Large (1)	4.4329e-67	A	Large (0)	7.3678e-71	B	Large (0)
10	4.3586e-67	B	Large (1)	7.4150e-61	A	Large (0)	6.9170e-65	B	Large (0)
15	3.8096e-66	B	Large (1)	1.2737e-60	A	Large (0)	4.0787e-64	B	Large (0)
20	2.7768e-66	B	Large (1)	1.5883e-60	A	Large (0)	2.5300e-64	B	Large (0)
25	3.8802e-66	B	Large (1)	2.1012e-60	A	Large (0)	2.5364e-64	B	Large (0)
30	1.7880e-66	B	Large (1)	4.6124e-60	A	Large (0)	4.8389e-65	B	Large (0)
35	2.0590e-65	B	Large (1)	2.1065e-59	A	Large (0)	8.6167e-65	B	Large (0)
40	4.2842e-65	B	Large (1)	3.8872e-59	A	Large (0)	8.7361e-65	B	Large (0)
45	1.1497e-64	B	Large (1)	5.5655e-59	A	Large (0)	1.5182e-64	B	Large (0)
50	2.5289e-65	B	Large (1)	6.5791e-59	A	Large (0)	2.7521e-65	B	Large (0)
55	2.7067e-65	B	Large (1)	6.9783e-59	A	Large (0)	2.7656e-65	B	Large (0)
60	6.6085e-65	B	Large (1)	7.0589e-59	A	Large (0)	6.6550e-65	B	Large (0)
65	2.0310e-65	B	Large (1)	7.0735e-59	A	Large (0)	2.0442e-65	B	Large (0)
70	2.5900e-64	B	Large (1)	7.0848e-59	A	Large (0)	2.6069e-64	B	Large (0)
75	2.0442e-65	B	Large (1)	7.1193e-59	A	Large (0)	2.0473e-65	B	Large (0)
80	6.6715e-65	B	Large (1)	7.1305e-59	A	Large (0)	6.6740e-65	B	Large (0)
85	2.0099e-64	B	Large (1)	7.1563e-59	A	Large (0)	2.0099e-64	B	Large (0)
90	2.0550e-65	B	Large (1)	7.1592e-59	A	Large (0)	2.0508e-65	B	Large (0)
95	5.0251e-65	B	Large (1)	7.1754e-59	A	Large (0)	5.0233e-65	B	Large (0)



(a) Average number of faults found (y-axis) per number of test cases selected (x-axis) for the similarity partition-based test case selection approach and a random selection.



(b) Average selection execution time (y-axis) per number of test cases selected (x-axis) for the similarity partition-based test case selection approach and a random selection.

Figure 6: Graphs comparing similarity partition-based test case selection and random selection.

Table 8: Data reported from Mann-Whitney U-tests and $\hat{A}_{12}$ effect size measurements when comparing similarity partition-based test case selection and random selection.]

(a) Mann-Whitney U-tests and $\hat{A}_{12}$ effect size measurements when comparing fault detection rate across the similarity partition-based test case selection approach and a random selection.

Comparison / Sample size (%)	Evolutionary (A), Random (B)		
	P-value	**Superior**	**Effect size**
5% - 80%	< 1.0e-50	A	Large (> 0.9200)
85% - 90%	< 1.0e-30	A	Large (> 0.7800)
95%	1.0217e-13	A	Medium (0.6371)

(b) Mann-Whitney U-tests and $\hat{A}_{12}$ effect size measurements when comparing selection execution time across the similarity partition-based test case selection approach and a random selection.

Comparison / Sample size (%)	Evolutionary (A), Random (B)		
	p-value	**Superior**	**Effect size**
5% - 80%	< 1.0e-64	B	Large (1)
85% - 90%	< 1.0e-65	B	Large (1)
95%	2.0134e-64	B	Large (1)

the faults, whereas similarity partition-based selections would reveal nearly 80% of the faults. This is a considerable efficiency gain in terms of early fault detection. The results show that similarity partition-based has a significantly better fault detection rate than the random approach for all sample sizes, with a large effect size, whereas the opposite is true for selection execution time. However, as long as the selection execution time is within a satisfactory range, the fault detection rate is obviously the most important criterion of the two. We can reasonably consider the selection execution time for similarity partition-based to be acceptable, particularly since we target smaller sample sizes of selection, in which the execution takes less than four minutes for a quite large original test suite. So, in order to address RQ4, similarity partition-based offers very strong advantages over the random approach when selecting test cases generated from classification tree models.

5.4. Discussion and Further Analysis

Although the idea of incorporating similarity-based test case selections within each partition seems intuitive and promising, especially given some of the recent results of its application in other contexts [3], the overall results we obtained are only marginally better when compared to a random selection within each partition. We analyzed the data more carefully to find plausible explanations for this unexpected result. It turns out that many of the faults are located

Selection strategy \ Sample size (%)	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75-100
Similarity Partition-Based	2%	5%	8%	13%	23%	29%	39%	53%	62%	68%	73%	78%	90%	97%	100%
Random Partition-Based	1%	3%	4%	10%	14%	17%	25%	33%	41%	47%	63%	73%	83%	95%	100%

Table 9: The average detection rate for Fault *Z* and *W* that are contained in a large partition. Each of the faults are present in one single test case among 340 test cases in the partition.

in partitions containing relatively few test cases. Hence, they would surface quickly with a partition-based approach, regardless of whether the selection was random or similarity-based within each partition. Such situations are expected to occur when testing is driven by an operational or usage profile, and faults are uncovered when executing unusual scenarios for which relatively few test cases are defined [20]. Though we have not used an operational profile, our original test suite is based on real taxpayer data, and consequently we have more test cases representing common scenarios that are well understood and less faulty, whereas we have fewer test cases accounting for more unusual situations where faults are more likely to hide. In situations when faults are only contained in small partitions, the impact of similarity-based selection within partitions becomes negligible and should not be applied, especially on very large test suites where selection execution time matters.

In our context, some of the faults were located in partitions containing many test cases. To be specific, 73% of the faults were located in small partitions only containing up to three test cases, whereas 27% of the faults were located in larger partitions of up to 340 test cases. As expected, the fault detection rate for the faults from the small partitions do not differ between similarity partition-based and random partition-based. So the main source of differences between random partition-based and similarity partition-based lies in the detection rate of the faults from the larger partitions. A detailed analysis shows that for all the larger partitions containing faults, the fault detection rate was significantly higher for similarity partition-based than random partition-based. And the larger the partition, the larger the impact of similarity measurement. As an example, fault *X* is contained in a partition with five test cases. In this case, similarity partition-based selection detected the fault in 112 out of 175 runs for the 5% selection sample, while the results for random partition-based was 111/175. For all other sample sizes, both selection strategies showed a 100% detection for this fault. This is a marginal improvement, which can be explained by the fact that the partition still contains few test cases. Fault *Y* is contained in a partition with 18 test cases. The partition is a bit larger than the previous one, and so is the fault detection improvement of similarity partition-based over random partition-based. For the 5% and 10% selection samples, the detection rate of fault *Y* is 31/175 vs. 23/175 and 99/175 vs. 98/175, both in favor of similarity partition-based, respectively. It is 100% for all other sample sizes. For the 5% sample, the difference is of practical significance. Faults *Z* and *W* are contained in the same partition constituting a total of 340 test cases. The fault detection rate for these faults combined are shown in Table 9. As the table shows, the similarity partition-based selection yields a better detection rate for all sample sizes, and the difference is practically significant for most sample sizes. By practically significant we mean that the relative increase in fault detection rate is considerable, i.e. 30% to 100% improvement up to 50% sample size. Table 10 also shows that the results are statistically significant (p-values are below our significance threshold) for most sample sizes with the effect size varying from small to large. The above results suggests that similarity partition-based should be preferred when faults are located in partitions containing a large number of test cases. This is often the case, for example, in safety critical systems where the most critical or complex scenarios and components tend to be more exercised by testing.

Another important point to highlight regarding the overall marginal improvement of similarity partition-based over random partition-based test case selection in this study, is the lack of variability in the subject model. To benefit significantly from similarity measurements when selecting test cases within partitions, the equivalence classes must be defined in such a way that they contain significant variation. If all model properties are defined as either having constant values (i.e. green, red or blue) or boolean values, there is little room left for test case diversity within each partition. In such cases all test cases within a partition would have the same encoding, and subsequently be equal, unless the distance function is expanded to define the distance between constants as well, e.g. blue is closer to green than red. Thus, an important prerequisite for using a similarity partition-based test case selection for classification tree models is to have at least one equivalence class for a property defined as a numerical range (i.e. 1–100 and > 100) or a string. The more diversity within partitions, the more likely our approach is to benefit from similarity partition-

Table 10: Mann-Whitney U-tests and $\hat{A}_{12}$ effect size measurements when comparing detection rate for fault Z and W across the similarity partition-based and random partition-based test case selection strategies.

Comparison / Sample size (%)	Evolutionary (A), Greedy (B)		
	p-value	**Superior**	**Effect size**
5	0.5586	A	Small (0.5057)
10	0.1757	A	Small (0.52)
15	0.0428	A	Small (0.5345)
20	0.0984	A	Small (0.5367)
25	0.0023	A	Small (0.5776)
30	7.8152e-05	A	Medium (0.6059)
35	3.7373e-05	A	Medium (0.6141)
40	1.6387e-09	A	Medium (0.6624)
45	2.4041e-10	A	Large (0.6715)
50	5.9040e-09	A	Medium (0.66)
55	0.0068	A	Small (0.5736)
60	0.2573	A	Small (0.5306)
65	0.0058	A	Small (0.5651)
70	0.2287	A	Small (0.5175)
75	NaN	None	NO effect (0.5)
80	NaN	None	NO effect (0.5)
85	NaN	None	NO effect (0.5)
90	NaN	None	NO effect (0.5)
95	NaN	None	NO effect (0.5)

based test case selection. The subject model in our study contained 11 properties at the bottom level, split up into 26 equivalence classes, 21 of them being constants, whereas 5 were defined as integer ranges. The magnitude of the ranges in equivalence classes were limited, i.e. 2–3, 2–5 and 2–9. Despite the limited variation enabled within the partitions, the results show that the fault detection rate improved (though to a limited extent) for all sample sizes when using similarity partition-based test case selections. The results were not significant for all sample sizes, nor was the effect size constantly large, but still the average fault detection rate was better for all sample values until a 100% detection rate was reached. Though more studies are required, our results suggest that a classification tree model accounting for even more variability than in our case study, would be likely to yield further benefits from a similarity partition-based test case selection.

To summarize the discussion, the level of variability and number of test cases within partitions are two important factors affecting the extent of the benefit that can be expected from similarity partition-based test case selection. In general, when modeling large complex systems in an industrial setting, it is likely that the equivalence classes defined in the model capture a variety of possible system values, as the model is a high level representation of the system. Consequently, many equivalence classes in the model will represent a range of possible system values and induce a great deal of variability within each partition. When executing test cases based on live system data, the distribution of test cases across partitions will vary, but many partitions are likely to contain large numbers of test cases. We recommend similarity partition-based test case selection under these conditions, while simpler solutions, such as random partition-based, are sufficient otherwise. One could also combine the two by defining a threshold value on the number of test cases per partition, and only use similarity based selection above this threshold.

5.5. Threats to validity

The fact that the study only includes one test suite derived from one classification tree model is a threat to the external validity of the study. Ideally, we should have included one or more additional test suites in the experiment, while varying model complexity and variability. But running and evaluating large test suites in real industry projects is a comprehensive and costly operation and the one test suite used in our study is large and based on operational data. It also triggers failures based on real faults detected in an operational system.

Throughout the experiment we compared several algorithms, by conducting multiple pairwise comparisons. This inflates the probability of a Type I error (reject the null hypothesis when it is true), which is a threat to conclusion validity. This could be adjusted by using, for example, *Bonferroni adjustment*. However, as reported by Arcuri and Briand (2011) [18], the Bonferroni adjustment has been repeatedly criticized in the literature. For example a serious problem associated with the standard Bonferroni procedure is a substantial reduction in the statistical power of rejecting an incorrect null hypothesis. So rather than performing adjustments, we have instead chosen to report all

p-values from all the statistical tests. The results are thus transparent and the readers have all the data at hand to form an opinion for themselves and to verify the conclusions we drew from our tests.

6. Conclusion and future work

Within the context of regression testing, we proposed a new strategy for selecting test cases generated from classification tree models, a well-known test generation strategy for black-box testing. Such an approach is a particularly useful alternative in an environment where source code analysis is either not convenient, scalable, or even possible. In short, our selection strategy selects, in a balanced way, test cases from all input partitions defined by the classification tree, while attempting to select the most diverse test cases from each partition. We conducted an experiment in an industrial setting–a large and critical database application developed by the Norwegian tax department–to determine which similarity-based selection algorithm, i.e., similarity function and selection algorithm, fared best for selecting test cases in a large regression test suite. We also compared our approach (coined similarity partition-based selection) against pure random selection and random selection within partitions. We compared both fault detection rate and selection execution time. In general random selection is superior to similarity-based selection in terms of selection execution time. However, the difference for smaller sample sizes in the range of interest is less than a few minutes (i.e., 39 seconds when selecting 30% of the test suite when comparing similarity partition-based with random selection). Given the limited practical consequences of the difference in selection execution time, we have used fault detection rate as the main criterion for comparison.

Among all alternatives, combining the Mahalanobis similarity function and the 1+1 EA algorithm proved to be the most efficient with regards to fault detection rate. The experiment also assessed whether similarity-based selection is a worthwhile and practical alternative to simpler solutions. Similarity partition-based test case selection offers far better fault detection rates compared to a random selection of test cases. For example, by selecting 5% of the test cases in a test suite, the fault detection rate of similarity partition-based is nearly 80%, as opposed to 25% for random selection. Despite a dramatical improvement over random, applying similarity-based selection within each partition only marginally improves, on average, the fault detection rate over random selection within each partition. Though similarity partition-based selection has better average fault detection rates for all sample sizes compared to random partition-based selection, the results are overall not statistically significant for all sample size values, and the measured effect size is in general low (i.e., low practical significance). The two most plausible explanations for these results are that (1) many of the faults in our study are located in partitions containing few test cases, along with the fact that (2) the subject classification tree model is such that diversity is often low within partitions, thus limiting the potential benefit of similarity-based test case selection. The results thus clearly suggest conditions under which similarity-based test selection is worth combining with a partition-based strategy. Consistent with such explanations, a more detailed analysis clearly shows that, for faults within large partitions, fault detection rates are significantly higher when using similarity over random within partition-based selection. This suggests that it would be beneficial to use similarity-based selection within partitions when these conditions are present, even when the increase in fault detection rate is obtained at the expense of higher execution time.

A possible future addition to this study could be to incorporate impact analysis to account for changes in the classification tree models when selecting a test case for regression testing. In our context this means identifying which partitions are affected by the change, and limit the testing to those partitions exclusively, or give them higher priority. Such an extension would not change the results reported in this study, since we would still use the same techniques within each partition. It is, however, an activity that would come with an additional cost, and whether the cost of performing an impact analysis would be worth the gain in improving regression test selection remains to be investigated. On the other hand, it would most likely improve scalability since it would allow us to focus on a subset of partitions.

References

[1] S. Yoo, M. Harman, Regression testing minimisation, selection and prioritisation: A survey, Software Testing, Verification and Reliability 22 (2012) 67–120.

[2] E. Lehmann, J. Wegener, Test case design by means of the CTE-XL, in: Proceedings of the 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000).

[3] H. Hemmati, A. Arcuri, L. Briand, Achieving scalable model-based testing through test case diversity, ACM Transactions on Software Engineering and Methodology (TOSEM) 22 (2013).
[4] F. G. O. N. Emanuela G. Cartaxo, Patrícia D. L. Machado, On the use of a similarity function for test case selection in the context of model-based testing, Software Testing, Verification and Reliability 21 (2011) 75–100.
[5] E. Rogstad, L. Briand, E. Arisholm, R. Dalberg, M. Rynning, Industrial experiences with automated regression testing of a legacy database application, in: 27th IEEE International Conference on Software Maintenance (ICSM), pp. 362–371.
[6] H. Hemmati, L. Briand, An industrial investigation of similarity measures for model-based test case selection, in: Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering, ISSRE '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 141–150.
[7] H. Hemmati, A. Arcuri, L. Briand, Empirical investigation of the effects of test suite properties on similarity-based test case selection, in: Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation, ICST '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 327–336.
[8] D. Liparas, L. Angelis, R. Feldt, Applying the Mahalanobis-Taguchi strategy for software defect diagnosis, Automated Software Engineering Journal 19 (2012) 141–165.
[9] R. Cilibrasi, P. Vitányi, R. Wolf, Algorithmic clustering of music, Computer Music Journal 28 (2004) 49–67.
[10] M. M. Deza, E. Deza, Encyclopedia of Distances, Springer, 2009.
[11] E. F. Krause, Taxicab geometry: An adventure in non-Euclidean geometry, Dover Publications Inc., 1988.
[12] R. D. Maesschalck, D. Jouan-Rimbaud, D. Massart, The Mahalanobis distance, Chemometrics and Intelligent Laboratory Systems 50 (2000) 1–18.
[13] C. H. Bennett, P. Gács, M. Li, P. M. B. Vitányi, W. H. Zurek, Information Distance, IEEE Transactions on Information Theory 44 (1998) 1407–1423.
[14] R. Cilibrasi, Statistical Inference Through Data Compression, Rudi Cilibrasi, 2006.
[15] S. Droste, T. Jansen, I. Wegener, On the analysis of the (1+ 1) evolutionary algorithm, Theoretical Computer Science 276 (2002) 51–81.
[16] T. J. Ostrand, M. J. Balcer, The category-partition method for specifying and generating fuctional tests, Communications of the ACM 31 (1988) 676–686.
[17] J. Aoe, Computer algorithms: String pattern matching strategies, Practitioners Series, IEEE Computer Society Press, 1994.
[18] A. Arcuri, L. Briand, A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering, in: 33rd International Conference on Software Engineering (ICSE), pp. 1–10.
[19] A. Vargha, H. D. Delaney, A critique and improvement of the CL common language effect size statistics of McGraw and Wong, Journal of Educational and Behavioral Statistics 25 (2000) 101–132.
[20] J. D. Musa, Software reliability engineering, AuthorHouse, 2 edition, 2004.