Reducing the Cost of Model-Based Testing through Test Case Diversity

Hadi Hemmati[1,2], Andrea Arcuri[1], and Lionel Briand[1,2]

[1] Simula Research Laboratory
[2] Department of Informatics, University of Oslo
`{hemmati,arcuri,briand}@simula.no`

Abstract. Model-based testing (MBT) suffers from two main problems which in many real world systems make MBT impractical: scalability and automatic oracle generation. When no automated oracle is available, or when testing must be performed on actual hardware or a restricted-access network, for example, only a small set of test cases can be executed and evaluated. However, MBT techniques usually generate large sets of test cases when applied to real systems, regardless of the coverage criteria. Therefore, one needs to select a small enough subset of these test cases that have the highest possible fault revealing power. In this paper, we investigate and compare various techniques for rewarding diversity in the selected test cases as a way to increase the likelihood of fault detection. We use a similarity measure defined on the representation of the test cases and use it in several algorithms that aim at maximizing the diversity of test cases. Using an industrial system with actual faults, we found that rewarding diversity leads to higher fault detection compared to the techniques commonly reported in the literature: coverage-based and random selection. Among the investigated algorithms, diversification using Genetic Algorithms is the most cost-effective technique.

Keywords: Test case selection; Model-based testing; Search-based testing; Clustering algorithms; Similarity measure; Genetic Algorithms; Adaptive Random Testing; Jaccard Index; UML state machines.

1 Introduction

The idea of model-based testing (MBT) [1] is to generate executable test cases by systematically analyzing specification models (e.g. represented as UML state machines) following a test strategy such as a coverage criterion, that aims to cover certain features of the model (e.g., all transitions). MBT brings many advantages but also entails the additional cost of modeling the software under test (SUT). In addition, there are two factors that significantly increase the cost of MBT: (1) the lack of automated oracle (e.g., when assessing the subjective perception of a media quality in a videoconference system), and (2) the high cost of test case execution (e.g., when testing must be performed on actual hardware or a restricted-access network). In both situations, the test suite must be as small as possible while, to the extent possible, preserving its fault revealing power. However, for real world size models, MBT techniques usually generate large sets of test cases regardless of the applied coverage criteria. Therefore, a model-based technique is required to select an optimal subset of test cases to be executed, which is, in general, a NP-hard problem.

In similarity-based test case selection, the idea is to diversify the selected test cases with respect to a similarity measure. In [2, 3], we proposed a similarity-based selection technique for testing based on UML state machines (SMBT). We compared different similarity measures in terms of what information from the test cases they have to evaluate (test case encodings) and how this evaluation should be done (similarity functions). The results showed that, in the context of SMBT, the similarity measure that represents a test case as a set of trigger-guards [2] and uses Jaccard Index [4] as the similarity function [3] is the most effective measure in terms of fault detection rate (FDR).

In this paper, we take a deeper look into the idea of diversifying test cases and investigate why similarity-based selected test cases are effective in finding faults. We also study different strategies that, given a similarity measure and a test suite, we can use to select a subset of the test suite. We applied our experiments on an industrial software system and a set of actual faults, and the results clearly showed that rewarding diversity is effective. The main explanation is that the test cases that find different faults belong to distinct clusters based on the similarity measure. In addition we found that, among different selection strategies, Genetic Algorithms (GAs) [5] are the most cost-effective technique for similarity-based test case selection. We also have shown that, in our case study, we could save up to 80% of test case executions, and get more than 99% FDR, by using a GA compared to a coverage-based selection technique.

The rest of the paper is organized as follows. Section 2 introduces the similarity-based test case selection technique. Section 3 discusses the different strategies which are used in this paper to diversify test cases. Section 4 provides a brief overview of related works covering similarity-based selection techniques. Section 5 reports the experimentation results of applying the selection techniques on an industrial case study. Section 6 concludes the paper and outlines our future work plan.

2 Similarity-Based Test Case Selection

Unlike coverage-based selection, where the goal is maximizing the coverage of the test model by the selected test cases (e.g. transition coverage in SMBT) to maximize chances of fault detection, similarity-based selection techniques maximize diversity among the selected subset. Diversity is calculated using a (dis)similarity measure between pairs of test cases. A similarity measure is the value that a similarity function assigns to two inputs which are being compared. In a testing context, inputs are usually encoded as a set or sequence of elements. In the context of MBT, the inputs are abstract test cases defined on the test model rather than concrete test cases. We do not use the execution information of the test case as, in our context, the goal is to select them before execution. Abstract test cases are naturally generated as a first step by MBT and can hide the unnecessary information for similarity comparisons. For example, in SMBT an abstract test case representation can be a path in the state machine specifying the SUT. In general, different faults can be detected by the same test path instantiated with different test data (e.g., event parameter values). Therefore, to compare different techniques, it is necessary to run the selected test paths with different input data and analyze their FDR distribution.

Representation (encoding) of the test cases has an important effect on the similarity measure. Though in SMBT a test path represents an encoded abstract test case, the test path can be described at different levels of details. In [2], we studied three encodings for a test path in UML state machine: state-based, transition-based, and trigger-guard-based, and reported that trigger-guard-based encoding is the most effective one in terms of fault detection, where a test path(*tp*) is represented as:

$$\begin{aligned} &\text{<tp>} &&::= \text{<TrGu>} \mid \text{<TrGu>} \text{ “,” } \text{<tp>} \\ &\text{< TrGu >} &&::= \mathit{trig} \mid \mathit{guard} \mid \mathit{id} \mid \mathit{guard} \text{“+”} \ \mathit{trig} \end{aligned}$$

where *trig* is the identification of a trigger, and *guard* is the identification of a guard in the state machine. In this representation, a transition is identified by its trigger and guard. It can be only a trigger, or a guard or both together. If there is a transition with no guard and trigger, we use the transition id (*id*) as its identifier.

Given an encoding, one may use different similarity functions to calculate the similarity value. In [3] we studied different set-based and sequence-based similarity functions and proposed Jaccard Index as the most cost-effective. Given a set of *n* encoded test cases (s_n) and a similarity function (*SimFunc*), the test case selection problem is reformulated as minimizing $SimMsr(s_n)$:

$$SimMsr(s_n) = \sum_{tp_i,\, tp_j \in s_n \,\wedge\, i>j} SimFunc(tp_i, tp_j)$$

where $SimFunc(tp_i, tp_j)$ returns the similarity of two test paths (or other encoded abstract test cases in MBT) in s_n represented by tp_i and tp_j. The last step in similarity-based selection is using a strategy to select a subset of test cases with minimum average pair-wise similarity (*SimMsr*). In the rest of this paper, we focus on finding the best strategy for this selection.

3 Strategies for Maximizing Diversity

Given a similarity measure we have two strategies to select the most diverse test cases. One is based on clustering test cases and taking samples from each cluster and the second is searching for the most diverse subsets. In this section, we introduce one clustering and two search techniques that will be investigated.

3.1 Clustering-Based Techniques

Clustering algorithms divide data instances into natural groups by maximizing their internal homogeneity and external separation [6]. Regardless of the specific algorithm which is used for clustering, most clustering techniques use a proximity measure as a mean to determine the closeness (similarity), or dissimilarity (distance) between pairs of instances and pairs of clusters.
In this study, we are using one of the simplest clustering algorithms, which has been frequently used in software engineering, including software testing [7]: Agglomerative Hierarchical Clustering (AHC) [6]. AHC starts with forming clusters containing

each exactly one object (a test case in this study). A sequence of merge operations is then performed until the desired number of clusters is achieved. At each step, the two most similar clusters will be joined together. The measure that we used for assessing similarity between two clusters, inter-cluster similarity, is Average Linkage and it is defined as the average of all pair-wise similarities between all instances of those two clusters [6]. After applying clustering, we need a sampling technique for selecting one or more test case per cluster. We use one-per-cluster sampling where the number of clusters is the same as the selected sample size and then randomly select one member from each cluster. The pseudo-code of the employed AHC follows:

```
(1) Make one cluster ($C_k$) per test path ($tp_i$).
(2) While the number of clusters is more than $sampleSize$
(3) Find the two most similar clusters $C_x$ and $C_y$ (with the maximum
    $InterClusterSim(C_x, C_y)$). Where:
```

$$InterClusterSim(C_x, C_y) = \frac{\sum_{tp_i \in C_x \wedge tp_j \in C_y} SimFunc(tp_i, tp_j)}{|C_x| * |C_y|}$$

```
(4) Merge the two clusters.
```

3.2 Test Case Selection Using Adaptive Random Testing

Another technique that we investigate is Adaptive Random Testing (ART), which has been proposed as an extension to Random Testing [8]. Its main idea is that diversity among test cases should be rewarded, because failing test cases tend to be clustered in contiguous regions of the input domain. This has been shown to be true in empirical analyses regarding applications whose input data are of numerical type [8]. Recently, Object-Oriented software has been also shown to manifest such a property [9]. Therefore, ART is a candidate selection strategy in our context as well. In this paper, we use the basic ART algorithm described in [8], but we ensure that no replicated test case is given in output. The pseudo-code for ART is:

```
(1) Z={}
(2) Add a random test case to Z
(3) Repeat until |Z|= $sampleSize$
(4)     Sample K random test cases that are different from Z
(5)     For each of these test cases k
(6)             k.maxSim = max(SimFunc(k , z ∈ Z))
(7)             Add the k with minimum maxSim to Z
```

3.3 GA-Based Test Case Selection

A GA [5] is used in this paper since the nature of our problem, which is a form of optimization, resembles typical problems addressed in search-based software engineering [10]. GAs are the most used and successful reported technique in this domain [10] and rely on four basic features: population, selection, crossover and mutation. More than one solution is considered at the same time (population). At each generation (i.e., at each step of the algorithm), some good solutions in the current population, selected by the selection mechanism, generate offspring using the crossover operator. This operator combines parts of the chromosomes (i.e., the solution representation) of the offspring with a certain probability; otherwise it just produces copies of the parents. These new offspring solutions will fill the population of the next generation. The mutation operator is applied to make small changes in the chromosomes of the offspring. Eventually, after a number of generations, an individual that solves the addressed problem will be evolved.

In this paper, we use a steady state GA as a selection technique, in which only the offspring that are not worse than their parents are added to the next generations. An individual in our context is a subset of size *n* from the original test suite (denoted s_n). Given a similarity function $SimFunc(tp_i, tp_j)$, the fitness function *f* to minimize is the sum, for all pairs (tp_i, tp_j) in s_n, of $SimFunc(tp_i, tp_j)$, denoted *SimMsr*. We use a single point crossover with probability of P_{xover} to combine two different parents s_n^x and s_n^y. A mutated test path is replaced by a test path that is selected at random from the set of all possible test paths. A valid solution is a set of test cases in which there is no duplicate. We have applied two types of stopping criteria for the GA in this study: (1) stopping after specific number of fitness evaluations, and (2) stopping after a fixed period of time (e.g., 350ms). The pseudo-code of the employed GA follows:

```
(1) Sample a population G of $m$ sets of test cases uniformly from the search space
    (i.e., the set of all possible valid sets with a given size $n$)
(2) Repeat until the stopping criterion is met
(3)     Choose $s_n^x$ and $s_n^y$ from G
(4)     $(\acute{s}_n^x, \acute{s}_n^y) :=$ crossover $(s_n^x, s_n^y, P_{xover})$
(5)     Mutate$(\acute{s}_n^x, \acute{s}_n^y)$
(6)     If valid $(\acute{s}_n^x, \acute{s}_n^y) \wedge \min(f(\acute{s}_n^x), f(\acute{s}_n^y)) \leq \min(f(s_n^x), f(s_n^y))$
(7)     Then $s_n^x := \acute{s}_n^x$ and $s_n^y := \acute{s}_n^y$
```

4 Related Work

There are three approaches reported in the literature to select a subset of test cases from a test suite that can be applied in our context: (1) Random or semi-random selection [11], where there is no guidance to select test cases; (2) Coverage-based selections, where we hypothesize that "the test cases which have more coverage are more likely to detect faults" (e.g., in [12] a Greedy search selects, at every step, the test case that covers the most uncovered statements whereas in [13, 14] a GA is used to find the maximum coverage.); (3) Similarity-based selections try to diversify test cases, given a similarity measure, assuming that maximizing diversity among the selected test cases maximizes the number of detected faults.

Diversifying test cases has been studied on code-based test case selection and mostly in the context of regression testing. The similarity measure in such cases is usually based on code coverage [7, 15-18]. In [19] a sequence of memory operations is used to calculate the similarity and in [20] the authors use the whole test script in string format as the input for similarity function. The work in [21] is the only one where the similarity function is based on model-level information. Test cases are represented as sequence of transitions in a LTS model of the system and the number of identical transitions in the sequence is the similarity function. Our similarity measure is different from theirs, both in terms of encoding and similarity function—we use trigger-guard sets on UML state machines and apply the Jaccard Index. In [2, 3] we have compared the effectiveness of our similarity measure with the measure in [21] and the results showed a great improvement using our technique, which therefore is applied in this study as well. Given a similarity measure, different strategies have been used to diversify the selected subsets: Greedy search in [17, 19-21], Neural network based classification in [18], ART in [17], AHC in [7, 15, 16]. In this paper, using our similarity measure, we compare ART, AHC, and a GA.

5 Empirical Evaluation

5.1 Case Study Description

The SUT under study is a typical safety monitoring component in a safety-critical control system implemented in C++ and modeled as UML state machines complemented by constraints specifying state invariants and guards. This SUT is typical of a broad category of reactive systems interacting with sensors and actuators. The first and the subsequent maintained versions of the system (including models and code) were developed and verified by company experts and our research team. The correct and the most up-to-date UML state machine, representing the latest version of the SUT's behavior, consists of one orthogonal state, 17 simple states, six simple-composite states, and a maximum hierarchy level of two. The unflattened state machine contains 61 transitions and the flattened state machine consists of 70 simple states and 349 transitions.

The correct latest UML state machine was given to our test case generation tool (TRUST) [22] as an input model. Using All-Transitions coverage, 281 test paths and corresponding executable test cases were automatically generated. In our case study, if a test path has the ability to detect a fault, it can be detected by any valid test data for that test path. Therefore, in our experiment, we have one test case per test path and the FDR of a test path is equal to the FDR of the corresponding test case. As it is typical in many embedded systems, the average execution time for these test cases is in the order of minutes, which makes running all the 281 test cases very time consuming.

We use 15 faulty versions of the code that are made by introducing one real fault per program. The 15 faults used in the study were introduced during maintenance activities by developers and re-introduced for the purpose of the experiment in the latest version of the SUT. Each of these faults belongs to one of the following categories: wrong guards on transitions, wrong state invariant, missing transition, and wrong OnEntry action in states. Among 281 test cases, 207 cannot detect any faults and 74 catch at least one fault. The average number of detected faults per test case for the 15 faulty versions is 0.72 and the maximum is five. Each fault is also detected on average by 13 test cases. There are nine faults which are only detected by three test cases and two faults are detectable by 65 test cases.

5.2 Experiment Design

In our industrial case study, we investigate the following research questions:

- **RQ1.** Why does diversifying test cases improve fault detection?
  - **RQ1.1.** Do test cases that find the same faults tend to be more similar to each other than with other test cases?
  - **RQ1.2**. Do test cases that find different faults tend to be more different from each other than test cases that find the same faults?
- **RQ2.** What is the most cost-effective way to diversify (given our similarity measure) a set of test cases?
  - **RQ2.1.** Does clustering-based test case selection improve the average FDR compared to coverage-based and random selection?
  - **RQ2.2.** Are search-based techniques more cost-effective than clustering-based selection in terms of fault detection?
- **RQ3.** How cost-effective is diversifying test cases compared to state of practice techniques for test case selection?

In RQ1 we are analyzing why diversifying test cases improves FDR. In other words, are test cases distinctly clustered with respect to different faults? We have carried out an exhaustive analysis based on our industrial case study. Given N=281 test cases, we ran all of them on the actual SUT and all its faulty version to check which of the M faults they are able to detect (in our case study M=15). We then calculated the similarity of each pair of test cases, for a total of N*(N-1)/2 pairs. Note that the exhaustive analysis of the search space landscape is based on the similarity values of all test case pairs. However, test case selection is performed for any arbitrary *sampleSize* where using an exhaustive search is not an option, since the search space size for selecting a subset of size *sampleSize* is equal to the number of possible *sampleSize* combinations within a test suite of a given size. In our case, as an example, the search space size for *sampleSize* =28 (~10% of the test suite) is 2.9*10^38.

To address RQ1, we investigate two hypotheses: (1) For each fault cluster, the similarity between pairs of test cases that find the same faults is, on average, significantly higher than the similarity of other test case pairs in the test suite, and (2) For each pair of fault clusters, the similarity between test cases that find different faults is significantly lower than the similarity of test case pairs that find the same fault in the test suite. If hypothesis (1) holds, then test cases finding the same faults will cluster in close areas of the test case space. As a result, rewarding diversity in test case selection would be beneficial. But hypothesis (2) should also hold, otherwise diversity might be harmful since we would need more than one test case from the same area to detect all faults.

In RQ2, we are interested in how to diversify the test cases, given the similarity measure used in RQ1. Our baselines of comparison are random selection (Rnd) and a coverage-based selection technique (CovGr) which is based on one of the most used selection techniques in the literature: it applies a Greedy search to maximize the coverage of the selected test cases [12]. In this paper, in each step of the Greedy search in CovGr, we look for the test cases which cover the most yet uncovered transitions on the UML state machine representing the SUT. Finally, in RQ3 we look at the practical benefits of our proposed approach based on our industrial SUT. In this study, as mentioned in Section 3, AHC is used as our clustering algorithm and a GA and ART as search-based techniques. Our measure of effectiveness is the FDR of the selected subset from the original test suite. Ideally, given the same amount of computational cost, we would say that a technique is better than the other if it obtains higher average FDR. For practitioners, such cost would typically be measured as the time that an algorithm takes before completing its task. Comparing algorithms using time is not a robust option from a practical standpoint though. Low-level implementation details may have a strong effect on computational time. If we use time as stopping criterion, then we may not truly compare algorithms but instead their implementations [23]. To cope with this problem, a measure that is independent from implementation details would be useful. For example, when comparing search algorithms, it is a common practice to allow each algorithm to run until a maximum number of fitness evaluations is executed (e.g., 100,000 [24]). However, the assumption here is that the total search cost is proportional to the number of fitness evaluations and the cost of other operations than fitness evaluation is either equal or negligible in both algorithms.

To compare GAs with ART, following the same general reasoning, we use the number of similarity comparisons (C) as stopping criterion, where n is the size of the output test case set. We hence can run both the GA and ART with the same preset number of similarity comparisons. For a GA that runs for W fitness evaluations (each consisting of Q similarity comparisons), we have that $C(GA) = W * Q = W * n * (n-1)/2$ whereas for ART we have [8]: $C(ART) = K * n * (n-1)/2$.

We would like to run both ART and the GA such that C(ART)=C(GA), but that might not be possible because K (the size of the candidate set in ART) is a constant that is upper bounded by N (281 in our case). In other words, the basic ART cannot be run for an arbitrary amount of computational resources as it is the case for GAs (for which we can choose arbitrarily high values for W). To cope with this problem, we can just run ART several independent times (e.g., J times), and then take the best result out of these J runs. Therefore, to obtain fair comparisons using similarity measures, we can simply enforce W=J*K.

Whenever we could not use a fair metric (as the number of fitness evaluations) for comparing different algorithms for test selection, we used the time expressed in milliseconds as stopping criterion, which is the time spent by our implementation of the algorithms on a PC with Intel Core(TM)2 Duo CPU 2.40 Hz and 4 GB memory running Windows 7. As we previously discussed, though this is not particularly robust in general, it is a reasonable option in our context as a significant effort was made to optimize implementations and the execution environment was stable.

To account for the randomness of the results, which exists for all selection algorithms, we ran each experiment 100 times and analyzed distributions. We report the results for different techniques for sample sizes less than 140 (~50% of the test suite) with intervals of 10, since our focus is, for practical reasons, on smaller size subsets. (In practice, test case selection is mostly used for selecting a relatively small sample of large test suites.) Furthermore, for large sample sizes, all selection techniques will usually be as good as random selection and typically detect most faults. We have performed non-parametric (Mann-Whitney U-test) statistical tests, using a significance level of 0.05, to compare the FDR distribution of the proposed and alternative selection techniques. Non-parametric tests are more robust than a parametric test (e.g., the *t*-test) when there are strong departures from normality and for large enough samples, as this is the case in this study (100 observations).

5.3 Experiment Results

5.3.1 Why Does Diversifying Test Cases Improve Fault Detection?

For each of the M=15 faults, we calculated the similarity of the test case pairs that both found each of these faults (groups of test case pairs, from F1 to F15). Mann-Whitney U-tests were performed (α =0.05) to see whether there was a difference in similarity value between the pairs in F1 to F15 and the set of all remaining pairs of test cases (T - Fi). Table 1 summarizes the results where bold median values represent statistically significant differences between the distributions of these Fi with T - Fi. Note that F1 and F2, F3 and F4, and F7 to F15 are on the same table row, as they have the same descriptive statistics. This is due to the fact that most test case pairs are the same and those that are not the same have high similarity values (according to our similarity measure).

The results show that the difference is significant for the first six groups. The other groups also show a high difference in terms of mean and median but, since there are only three observations for each of those groups, we cannot get statistically significant differences. Therefore, the first hypothesis of RQ1.1: “Test cases that find the same faults tend to be more similar to each other than with other test cases” is confirmed.

To investigate RQ1.2, for each pair of fault clusters Fi and Fj, let us consider the similarity distribution (Dd) of test case pairs which belong to two different clusters, i.e., test cases that find different faults. We compare Dd with the similarity distribution (Ds) of test case pairs which both are in one of those two clusters, i.e., test cases that find the same fault. The median of Dd and Ds per cluster pair is reported above the diagonal in Table 2.

There are cases where fault clusters Fi are exactly the same, i.e., their respective faults are found by exactly the same set of test cases. Distinguishing them does not have any effect on the FDR results (either all or none of the faults will be revealed by a selected set of test cases) and therefore such clusters are not distinguished. As a result, there are seven distinct fault clusters (labeled as A to G) matching the columns and rows of Table 2. Their mapping to the 15 fault clusters is as follows: A(F1 and F2), B(F3 and F4), C(F5), D(F6), E(F7 to F9), F(F10 to F12), G(F13 to F15).

The bold values show the cases where there is a statistically significant difference between Dd and Ds, based on a Mann-Whitney U-test. The presence of significant differences support the claim that fault clusters are far away from each other and therefore that rewarding diversity is useful. In cases where two clusters are overlapping, the size of the overlap compared to the size of their union will determine whether rewarding diversity is harmful. If the ratio of the overlapping part (intersection) over the union is high, a test case that finds one of the two faults would have a high probability of finding the other. In this case, rewarding diversity is still a reasonable option. We measure this ratio by dividing the size of two clusters' intersection |I| by the size of their union |U|: I_U=|I|/|U|. The cells below the diagonal of Table 2 report this measure per cluster pair.

Among 21 cluster pairs, 15 contain distinct clusters with significant differences between Dd and Ds. There are three clusters (E, F, and G) that only contain a few test cases (three per cluster), which are not amenable to statistical analysis and show no statistically significant differences. Clusters B and D which are not significantly different from each other show a high overlapping value (0.57), implying that although these clusters are not distinct, there is a 57% probability that a test case that is selected from their union can find both faults. Two cluster pairs, <A,D> and <C,D>, show unexpected results—Dd median lower than the Ds median—and they are not highly overlapping. Therefore, since among 21 pairs, 15 pairs fit the situation where similarity-based selection is effective, two do not, and four are neutral, we can conclude that, overall, in most cases "test cases that find different faults tend to be more different from each other than test cases that find the same faults".

Table 1. Min, max, median, mean, and standard deviation of similarity values of the test cases that find the same faults

Groups	Pairs	Min	Median	Mean	Max	SD
T	39340	0.076	0.250	0.291	1.000	0.166
F1,F2	2080	0.181	**0.4**	0.432	1.000	0.173
F3,F4	91	0.375	**0.571**	0.561	0.833	0.143
F5	28	0.200	**0.464**	0.475	0.800	0.168
F6	28	0.714	**0.714**	0.714	0.714	0.000
F7 to 15	3	0.375	0.428	0.434	0.500	0.062

Table 2. Each cell above the diagonal shows the median of Dd and Ds (Dd/Ds) and each cell below the diagonal shows the overlapping measure (I_U), per cluster pairs. Bold median values highlight significant differences (Mann-Whitney U-test) between the Dd and Ds.

	A	B	C	D	E	F	G
A	-	**0.33/0.42**	**0.33/0.40**	**0.71/0.40**	**0.18/0.40**	**0.18/0.40**	**0.18/0.40**
B	0.21	-	**0.37/0.57**	**0.71/0.66**	**0.37/0.57**	**0.37/0.57**	**0.37/0.57**
C	0.12	0	-	0.71/0.71	**0.37/0.42**	**0.37/0.42**	**0.37/0.42**
D	0.12	0.57	0	-	**0.11/0.71**	**0.11/0.71**	**0.11/0.71**
E	0	0	0	0	-	0.37/0.42	0.37/0.42
F	0	0	0	0	0	-	0.37/0.42
G	0	0	0	0	0	0	-

Overall, the results of our analysis confirm that diversity in test case selection should be encouraged and that our similarity measure is adequate. It also seems that since test cases finding the same faults are clustered together and these clusters are mostly distinct, clustering algorithms are a reasonable candidate approach to achieve diversity, though we will investigate what is the best strategy in the next research question.

5.3.2 What Is the Most Cost-Effective Way to Diversify a Set of Test Cases?

To answer RQ2 we first compare the AHC clustering algorithm with CovGr and Rnd introduced in Section 5.2. Fig. 1 shows the FDR results of the algorithms.

Overall, the results show that for all sample sizes AHC is more effective than its two alternatives except that for sample sizes less than 30 (~10% of the test suite) the difference between the average FDRs of CovGr and AHC is not statistically significant (based on Mann-Whitney tests). Considering the fact that in practice the results for smaller sample sizes are more important, AHC may not be preferred to CovGr given the high cost of a clustering technique compared to simple Greedy search. On average (for all sample sizes over 100 runs) each selection requires 350ms, 10ms, and less than 1ms when using AHC, CovGr, and Rnd, respectively. Though those time differences may not seem relevant, they may become so on much larger test suites of thousands of test cases. However, for sample sizes higher than 40, there is a huge (up to 40%) improvement using AHC compared to CovGr. In addition, AHC ensures 100% FDR with 80 test cases whereas CovGr and Rnd find less than 95% of the faults even with 140 test cases.

Note that, in theory, since Rnd does not use any heuristic to increase FDR, we cannot improve it. However, we can improve CovGr by running it several time with different random selections, wherever the coverage among alternative test cases is equal, and reporting the best result out of all runs. To compare the FDR results of CovGr when it costs exactly the same as AHC, we let CovGr improve its results by random reselection and stopped the algorithm after 350ms. The results showed that in our case, there is no practically significant difference in CovGr FDR for 10 and 350 ms of running time.

Addressing RQ2.1, given that the FDR of AHC is always equal or superior to that of CovGr or Rnd, and the fact that we cannot predict for a given test suite the sample size threshold above which AHC will be certain to fare significantly better, we favor the systematic use of AHC over CovGr and Rnd. Moreover, in practice, this strategy makes even more sense when considering that test case execution time (which in our case is in the range of minutes) is usually much higher than selection time for any of the techniques (which in our case is in the range of milliseconds).

Comparing search-based techniques with AHC, first we need to find out which search technique is more cost-effective. In this study, we compare the FDR of ART and a GA. The GA is stopped after 10,000 fitness evaluations, and ART is run 1000 times with K=10 (so both algorithms use the same number of similarity comparisons). Fig. 2 shows the average FDR of the techniques for each sample size over 100 runs. In general, the GA fares better and more particularly so from sample size 20 (~7% of the test suite) to 70 (~25% of the test suite). For sample sizes larger than 70, the FDR of both techniques converges to 1.0. The differences for smaller sample sizes are statistically significant but, because these differences may not practically significant (at most 10% improvement for the GA), we need to look closely at the relative cost of ART and the GA.

As we mentioned earlier, the number of fitness evaluations is usually a good platform-independent measure for the cost of search techniques. However, in our implementation, a matrix made of all pair-wise similarities is created before any search. Therefore, this overhead is the same for all search algorithms and the fitness evaluation is not an expensive part of the search. Therefore, we cannot be sure that total cost is proportional to the number of fitness evaluation. In Fig. 3, we have plotted the actual time spent by the two algorithms (ART and the GA with 10,000 fitness evaluations). The required time for 10,000 fitness evaluations using both techniques is exponentially increasing and they both spend almost the same time for very small sample sizes (less than 20). For sample sizes higher than 20 (~7% of the test suite), ART quickly gets more expensive than the GA. Given that it always has equal or worse FDR results, there is no reason for choosing ART over the GA.

In the next step, we compare AHC with the GA but using the same execution time that AHC requires for its selection (350ms). Fig. 4 shows that the GA is clearly preferable over AHC considering that spending the same time as AHC, the GA fares in general much better and can almost double the average FDR results of AHC for small sample sizes. It is also more effective in finding all faults: AHC requires 80 test cases whereas the GA only needs 40 test cases to achieve 100% FDR. To draw more conservative conclusions regarding the superiority of the GA, we even conducted another experiment and ran AHC a relatively long time (20,000ms) to compare its FDR result with the GA using 350ms (Fig. 4). However, even when letting AHC work for almost 60 times longer than the GA it still yields much lower FDR. Therefore, our suggested answer to RQ2 is using the GA over the other alternatives.

5.3.3 How Cost-Effective Is Diversifying Test Cases Compared to State of Practice Techniques for Test Case Selection?

To answer RQ3, we compare our best candidate based on RQ2, which is similarity-based selection using a GA, with a coverage-based Greedy search (CovGr). Looking at Fig. 1, the first observation is that the GA can save more than 80% of the test case execution cost, given the fact that the GA, on average, finds more than 99% of the faults by 40 test cases whereas CovGr requires more than 220 test cases to achieve the same (not plotted in the figure). To have a more detailed cost-effectiveness assessment, we look at the improvement that the GA may provide over time. Since this improvement varies over sample sizes as well, we plotted in Fig. 5 the percentage of FDR improvement provided by the GA over CovGr for five sample sizes: 10, 15, 20, 25, and 50 (ranging from 4 to 18 % of the test suite), over a time period of 10ms (the average time required by CovGr to select test cases) to 350ms (the average time required by AHC to select test cases). Note that as we mentioned earlier, as opposed to the GA, CovGr does not improve over time.

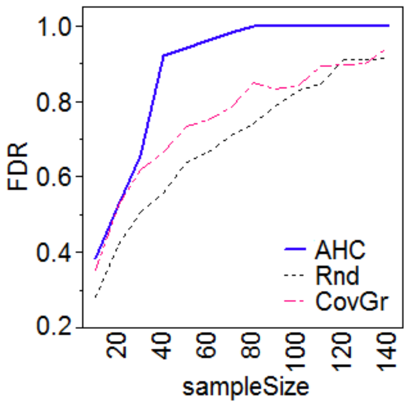

Fig. 1. The average FDR of AHC, Rnd, and CovGr for different sample sizes

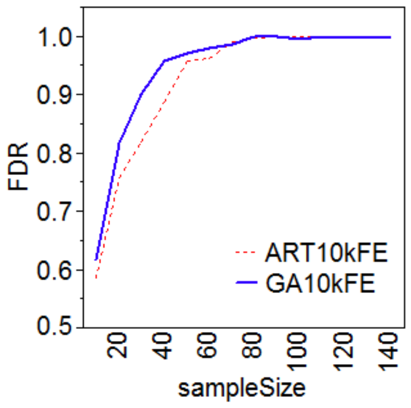

Fig. 2. The average FDR of ART and the GA with 10,000 fitness evaluations for different sample sizes

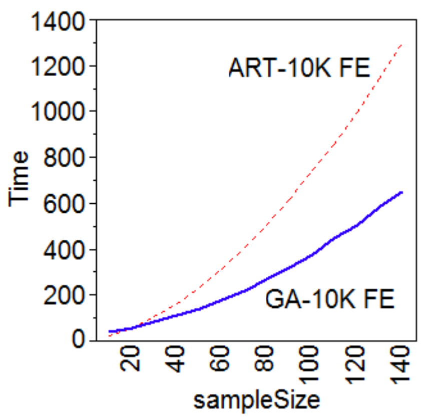

Fig. 3. The time in milliseconds required by the GA and ART to run 10,000 fitness evaluations for different sample size

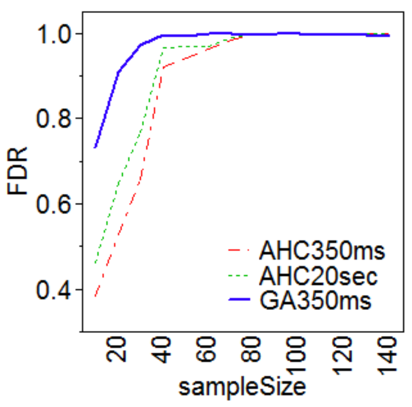

Fig. 4. The average FDR of the GA with 350ms and AHC with 350 and 20,000ms for different sample sizes

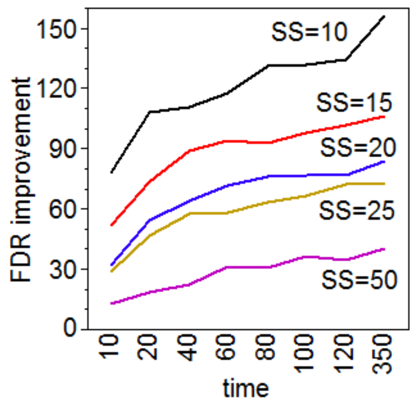

Fig. 5. The percentage of improvement of similarity-based selection using GA over CovGr for different sample sizes (SS) in time range of 10 to 350ms

A first observation from Fig. 5 is that the smaller the sample size, the larger the improvement provided by the GA. Also, it is interesting to see that the GA, even with 10ms execution time, always detects more faults than CovGr. For example, the average FDR of the GA is 80% larger than the CovGr FDR for 10ms. Finally, a cost analysis shows that in cases where we can afford spending more time for selection, the GA can be greatly improved. For all five sample sizes shown in Fig. 5, the GA's improvement over CovGr almost doubles if we give it 350ms instead of 10ms. This improvement over time gets very large when the sample size gets smaller. For example, for sample size 10 the GA can yield a 160% higher FDR than CovGr, which in practice is a great benefit given that the cost for this improvement is only 350ms for a test suite of 281 test cases where the cost of running one extra test case is in the order of minutes.

6 Discussion on Validity Threats

The main threats to the validity of this study are firstly the fairness of comparisons in terms of cost and secondly the generalizability of the results.

Similarity comparisons of test cases and clusters are the most influential part of selection techniques. In our implementations of the algorithms, all pair-wise similarities are pre-calculated in a similarity matrix which is given to the selection algorithm as an input parameter. Obviously, this implementation is not scalable and the similarity matrix will face memory limitations for large test suites. However, if we can afford pre-calculation, then the most expensive part of the search algorithms may not be the fitness evaluation anymore. We can see its effect on comparing ART and the GA where having the same number of similarity evaluations ART requires much more time. We have not studied on-demand similarity calculations, which might give different FDR results using the same stopping time. In addition, inter-cluster similarity calculation in AHC is very expensive and in our implementation it is repeated for each iteration of the algorithm. The code can be optimized by caching the similarities between clusters in each iteration and in the next iteration only calculate the similarities if it is not already available. However, implementing this improvement is not trivial since saving similarities of all combinations of clusters in all iterations may be not possible due to memory limitations. There is a tradeoff to be made between memory cost and execution speed.

The second issue is due to the fact that all our results and conclusions rely on a single industrial case study using a given set of real faults. Though running such studies is time consuming, it must obviously be replicated. However, as discussed earlier, the system used here is typical of a broad category of industrial systems: control systems with state-dependent behavior, controlling sensors and actuators.

7 Conclusion and Future Work

In practice, executing test cases generated by model-based testing (MBT) techniques is costly. This cost is due to the large test suites which are typically generated by MBT tools on industrial-scale systems to systematically achieve a coverage/adequacy criterion. However, for system level testing, in many situations testing should take place on the deployment platform where the cost (time and resource) of each test execution may be high. This may be due to the cost of using actual hardware, potential damages in case of failure, or access to restricted infrastructure (e.g., test network). In addition, for many systems, automatically generating oracles from models is very difficult or impossible. In such cases, test cases should be evaluated manually, greatly increasing the cost of test execution and analysis. In cases such as the ones mentioned above, one must execute a subset of the generated test suite whose size is dependent on context. In this paper, we propose a new approach for test case selection from UML state machines, by maximizing the diversity of the selected test cases. To measure diversity we used a specific test case representation for UML state machines (triggers-guards sets), which should be adapted in case of using other models, and a model-independent similarity function (Jaccard Index). We investigated why diversifying test cases with respect to our similarity measure increases fault detection rates and compared different strategies to diversify the test cases: Clustering, Adaptive

Random Testing, and Genetic Algorithms (GAs). The results of our study on an industrial software system and actual faults showed that: (1) rewarding diversity leads to finding more faults, (2) our proposed similarity-based selection (using Jaccard Index on the set of trigger-guards with a GA selection) is the most cost-effective approach compared to the other alternatives. In addition, we showed that in practice this approach can reduce the cost of test case execution in MBT by selecting a small set of test cases which can find all (or most) faults in short amount of time. In the future, we plan to replicate the study on another industrial system. In addition, we will evaluate alternative optimization and search techniques.

References

1. Utting, M., Legeard, B.: Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann, San Francisco (2006)
2. Hemmati, H., Briand, L., Arcuri, A., Ali, S.: An Enhanced Test Case Selection Approach for Model-Based Testing: An Industrial Case Study. In: 18th ACM International Symposium on Foundations of Software Engineering, FSE (2010)
3. Hemmati, H., Briand, L., Arcuri, A.: Investigation of Similarity Measures for Model-Based Test Case Selection. Simula Research Laboratory, Technical Report (2010-05) (2010)
4. Teknomo, K.: Similarity Measurement, `http://people.revoledu.com/kardi/tutorial/Similarity`
5. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, Reading (2001)
6. Xu, R., Wunsch II, D.C.: Survey of Clustering Algorithms. IEEE Transactions on Neural Netwoks 16, 645–678 (2005)
7. Yoo, S., Harman, M., Tonella, P., Susi, A.: Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In: 18th ACM International Symposium on Software Testing and Analysis, ISSTA (2009)
8. Chen, T.Y., Kuoa, F.-C., Merkela, R.G., Tseb, T.H.: Adaptive Random Testing: The ART of test case diversity. Journal of Systems and Software 83, 60–66 (2010)
9. Ciupa, I., Leitner, A., Oriol, M., Meyer, B.: ARTOO: Adaptive Random Testing for Object-Oriented Software. In: 30th IEEE International Conference on Software Engineering (ICSE) (2008)
10. Harman, M.: The Current State and Future of Search Based Software Engineering. In: Future of Software Engineering, pp. 342–357. IEEE Computer Society, Los Alamitos (2007)
11. Binder, R.V.: Testing Object-Oriented Systems: Models, Patterns, and Tools. Addison-Wesley Professional, Reading (1999)
12. Elbaum, S.G., Malishevsky, A.G., Rothermel, G.: Test Case Prioritization: A Family of Empirical Studies. IEEE Transactions on Software Engineering 28, 159–182 (2002)
13. Li, Z., Harman, M., Hierons, R.M.: Search Algorithms for Regression Test Case Prioritization. IEEE Transactions on Software Engineering 33, 225–237 (2007)
14. Ma, X.Y., Sheng, B.K., Ye, C.Q.: Test-Suite Reduction Using Genetic Algorithm. In: Cao, J., Nejdl, W., Xu, M. (eds.) APPT 2005. LNCS, vol. 3756, pp. 253–262. Springer, Heidelberg (2005)
15. Leon, D., Podgurski, A.: A Comparison of Coverage-Based and Distribution-Based Techniques for Filtering and Prioritizing Test Cases. In: 14th IEEE International Symposium on Software Reliability Engineering, ISSRE (2003)

16. Masri, W., Podgurski, A., Leon, D.: An Empirical Study of Test Case Filtering Techniques Based on Exercising Information Flows. IEEE Transactions on Software Engineering 33 (2007)
17. Jiang, B., Zhang, Z., Chan, W.K., Tse, T.H.: Adaptive random test case prioritization. In: 25th IEEE/ACM International Conference on Automated Software Engineering, ASE (2009)
18. Simão, A.d.S., Mello, R.F.d., Senger, L.J.: A Technique to Reduce the Test Case Suites for Regression Testing Based on a Self-Organizing Neural Network Architecture. In: 30th Annual International Computer Software and Applications Conference, COMPSAC (2006)
19. Ramanathan, M.K., Koyutürk, M., Grama, A., Jagannathan, S.: PHALANX: a graph-theoretic framework for test case prioritization. In: 23rd Annual ACM Symposium on Applied Computing (2008)
20. Ledru, Y., Petrenko, A., Boroday, S.: Using String Distances for Test Case Prioritisation. In: 24th IEEE/ACM International Conference on Automated Software Engineering, ASE (2009)
21. Cartaxo, E.G., Machado, P.D.L., Neto, F.G.O.: On the use of a similarity function for test case selection in the context of model-based testing. In: Software Testing, Verification and Reliability (2009)
22. Ali, S., Hemmati, H., Holt, N.E., Arisholm, E., Briand, L.: Model Transformations as a Strategy to Automate Model-Based Testing - A Tool and Industrial Case Studies. Simula Research Laboratory, Technical Report (2010-01) (2010)
23. Ali, S., Briand, L.C., Hemmati, H., Panesar-Walawege, R.K.: A Systematic Review of the Application and Empirical Investigation of Search-based Test-Case Generation. IEEE Transactions on Software Engineering, Special issue on Search-Based Software Engineering, SBSE (in press, 2010)
24. Harman, M., McMinn, P.: A Theoretical and Empirical Study of Search Based Testing: Local, Global and Hybrid Search. IEEE Transactions on Software Engineering 36, 226–247 (2010)