Test Suite Reduction based on Similarity of Test Cases

Ana Emília V. B. Coutinho[1,2], **Emanuela G. Cartaxo**[1], **Patrícia D. L. Machado**[1]

[1] SPLab-UFCG, Campina Grande, PB, Brazil

{ana, emanuela}@copin.ufcg.edu.br, patricia@computacao.ufcg.edu.br

[2]CCHE-UEPB, Monteiro, PB, Brazil

anaemilia@cche.uepb.edu.br

Abstract. *This paper presents a new strategy based on a similarity degree between test cases for test suite reduction in the context of Model-based Testing. The idea is to analyse the similarity degree among pairs of test cases and systematically remove them by starting from the most similar pairs while a set of test requirements can still be found. To evaluate the effectiveness of the strategy, we conducted an experimental investigation together with four other well-known heuristics presented in the literature, considering the measures of the reduced test suite size and failure coverage. Results show that the proposed strategy presents average reduction size along with a better failure coverage.*

1. Introduction

Model-based Testing (MBT) [Utting and Legeard 2007] approaches have been proposed with the aim to automatically generate test cases from specifications of the system behaviour. While MBT can reduce the effort to produce test cases, usually MBT approaches generate test suites that are too large, and maybe impossible to be executed completely due to resource constraints (time and money) [Bertolino et al. 2010].

To address this problem, approaches on how to reduce the size of test suites have been proposed in the literature, particularly, the test suite reduction approach (also known as *test suite minimization*) [Harrold et al. 1993]. Its goal is to define a subset from the original test suite that satisfies a given set of test requirements. A number of test suite reduction strategies for code level have already been investigated and experimented extensively in the literature [Rothermel et al. 2002]. These strategies are usually based on heuristics (the heuristics make the best local choice at each step with the goal of finding the best global) to maximize coverage, and the test requirements are defined usually as statement and decision coverage. However, different studies have shown that fault detection capability of test suites can be severely compromised by reduction [Rothermel et al. 2002]. On the other hand, investigation on reduction for specification-based test cases is recent, specially in the MBT context, with few strategies and experimental results. Furthermore, the results are still not conclusive and also divergent even in the code level context.

In this paper, we present a new strategy for test suite reduction based on similarity of test cases in the MBT context. The idea is to identify the degree of similarity between the test cases and keep in the suite the most different ones that together can find a set of test requirements. We focus on specifications that can be expressed as Labelled Transition Systems (LTS). To evaluate the strategy, an experimental investigation was conducted with different test suite reduction heuristics that can be applied in the MBT context. These strategies were analyzed by considering two metrics: i) reduced test suite size and ii) failure coverage. The inputs are 12 LTS specifications automatically generated with distinct configurations. For each specification, test cases were generated and a set of failures was randomly defined according to the test suite size (10%). This percentage of failures was defined according to average percentage of the real-world applications used in our research group in other studies. As test requirement, we considered *all-transition-pairs* coverage to capture different interactions between transitions and improve failure detection capability so that differences could be observed among the strategies. Results show that the new strategy can be more effective, by achieving a good balance between size and failure detection.

This paper is structured as follows. Section 2 defines basic concepts such as LTS models, test suite reduction, along with a short description of the heuristics considered in this work. In Section 3, we present our proposal to reduce test suites based on similarity. Section 4 presents the definition of the experiment goals and planning. Section 5 presents the results through the metrics collected and their statistical validity, and shows the analysis of these results. Section 6 discusses related work presented in the literature. Finally, in Section 7 we presented some concluding remarks.

2. Background

2.1. Labelled Transition System (LTS)

LTS is a directed graph defined in terms of states and labelled transitions between states to describe system behaviour. Formally, an LTS can be defined as a 4-tuple $\langle S, L, T, s_0 \rangle$, where: S: is a finite, non-empty set of states; L: is a finite, non-empty set of labels; T: is a subset of $S \times L \times S$ (set of triples), called the transition relation; and s_0: is the initial state, where $s_0 \in S$.

2.2. Test Suite Reduction

According to Harrold *et al.* [Harrold et al. 1993] the test suite reduction problem can be defined as follows:

Given: A test suite TS, a set of test requirements $Req = \{Req_1, Req_2, \ldots, Req_n\}$ to be covered, and subsets of TS: $TS_1, TS_2, \ldots, TS_n$, where each test case of TS_i can be used to test Req_i;
Problem: Find minimal subset $RS \subseteq TS$ that satisfies all of the ***Req's*** and has at least one test case for each *Req*.

In general, to find the minimal subset of the original test suite (Reduced Set - RS) that provides 100% test requirement coverage is NP-complete problem (minimization problems are NP-complete because they can be reduced to the *minimum set-covering* problem) [Cormen et al. 2001]. Below, we briefly described four well-known heuristics proposed for code-based reduction: Greedy, GE, GRE and H. These heuristics can also be applied on test suites obtained from MBT approaches.

Greedy Heuristic: it repeatedly selects the test case that satisfies the maximum number of unsatisfied test requirements [Cormen et al. 2001]. If there is a tie situation, an arbitrary choice is made. The selected test case is added to Reduced Test Suite (RS) and all test requirements that can be satisfied by that test case are marked as a satisfied test requirement. This algorithm stops when all test requirements are satisfied.

Heuristic GE: this heuristic is based on the essential concept [Chen and Lau 1998b]. A test case is *essential* when only this test case covers one specific requirement. Initially, all essential test cases are selected, and the test requirements satisfied by them are marked. Then, the greedy heuristic is applied.

Heuristic GRE: this heuristic, proposed by Chen *et al.* [Chen and Lau 1998a], is based on three strategies: the greedy heuristic, the 1-to-1 redundancy strategy, and the essentials strategy. A test case $t_{1-1} \in$ TS is said 1-to-1 redundant, if there is $t(\neq t_{1-1}) \in$ TS such that $Req(t_{1-1}) \subseteq Req(t)$, i.e., if the same set of requirements, that is covered by a test case, is also covered by other test cases, then those test cases are considered 1-to-1 redundant. The essentials strategy and 1-to-1 redundancy strategy are applied alternatively whenever possible. The greedy strategy is only applied if neither the essential nor 1-to-1 redundancy can be applied.

Heuristic H: Harrold et al. [Harrold et al. 1993] present a test suite reduction strategy, which we call *heuristic H*. The idea is to select test cases according to their degree of *essentialness*, i.e., keeping in the reduced set, the test cases in the order of most essential to least essential. Each test requirement has a requirement cardinality, that is the number of test cases that meet that requirement. First, the test requirement with cardinality one is considered. When a test case is added to the reduced set, all requirements covered by that test case are marked. Among the unmarked test requirements with lowest requirement cardinality, the algorithm selects the most frequently occurring test case. If there is a tie, the algorithm chooses the test case that occurs most frequently at the next highest requirement cardinality and so on (if there is a tie and the requirement cardinality is maximum, then the random choice is applied). This algorithm stops when the reduced set has test cases that cover all test requirements.

3. Similarity-based Test Reduction

Inspired by the similarity-based strategy for test case selection proposed by Cartaxo [Cartaxo et al. 2011], we defined our proposal for test suite reduction. On one hand, the goal of the selection strategy is to select a percentage of test cases that are the most different of the test suite based on the degree of similarity among them. On the other hand, reduction strategies, as said in Section 2.2, seek to produce a subset from the original test suite satisfying the same test requirements as the original suite. Hence, the goal is to minimally achieve coverage of a set of test requirements by the most different test cases. For this, the following inputs are necessary: i) *test suite:* the set of test cases that should be reduced; ii) *test requirements:* the set of requirements that should be covered; iii) *similarity matrix:* it presents the similarity degree for all the pairs of test cases.

To construct the matrix, it is necessary to calculate the degree of similarity between each pair of test cases. This similarity degree between two test cases can be measured from a distance function. In this work, we used an adaptation of the measure of redundancy proposed by Fraser and Wotawa [Fraser and Wotawa 2007]. The redundancy R of a test suite TS is defined with the help of the execution tree: $R(TS) = \frac{1}{n-1} \cdot \sum_{x \in childen(root(TS))} \mathcal{R}(x)$. According to Fraser and Wotawa [Fraser and Wotawa 2007], the redundancy of the tree is the ratio of the sum of the redundancy values $\mathcal{R}$ for the children of the root-node and the number of arcs in the tree ($n-1$, with n nodes). The redundancy value $\mathcal{R}$ is defined recursively as follows:

$$\mathcal{R} = \begin{cases} (|children(x) - 1|) + \sum_{c \in childen(x)} \mathcal{R}(c) & if \quad children(x) \neq \{\} \\ 0 & if \quad children(x) = \{\} \end{cases}$$

The strategy follows the Algorithm 1. Initially, the idea is to analyze all the values on the matrix from the highest value and verify whether its removal maintained 100% of the test requirements coverage. In summary, the `allValuesMatrixAnalyzed` method (line 1) analyzes the similarity degree of each test cases pair existing in the matrix. This method returns `true` if all values of the matrix were already analysed. Inside the repeating structure, the first step (lines 2 – 3) is to find the maximum value in the matrix. This value is associated to the two most similar test cases. When a tie among maximum values is found in the similarity matrix, one value is randomly chosen. In the second step (lines 4 – 5), the order of analysis of these two test cases is defined according to their path lengths. Then, the test case with the lower number of transitions is analyzed first. If these lengths are same, the order of the test cases is made randomly. In the next step (lines 6 – 13), it is necessary to verify if the reduced test suite satisfies all the requirements with the removal of the first test case chosen. If all the requirements are satisfied, the first test case chosen is also removed from the similarity matrix. Otherwise, the first test case is added again in the test suite, and then the other one (the second test case chosen) is removed from the test suite in a similar way. While all pairs of test case of the matrix are not analysed, new pairs of test cases continue to be selected, removed and tested in the similarity matrix

```
    input  : testSuite, testRequeriments, similarityMatrix
    output: reducedTestSuite
 1  while (!allValuesMatrixAnalyzed()) do
 2      maxValues = getAllMaxValue(similarityMatrix);
 3      pairs = maxValues.shuffle.get(0);
 4      firstChoice = pairs.getFirstTestCase();
 5      secondChoice = pairs.getSecondTestCase();
 6      if satisfyAllRequeriments(testSuite.remove(firstChoice)) then
 7          similarityMatrix.remove(firstChoice);
 8      else
 9          testSuite.add(firstChoice);
10          if satisfyAllRequeriments(testSuite.remove(secondChoice)) then
11              similarityMatrix.remove(secondChoice);
12          else
13              testSuite.add(secondChoice);
14  reducedTestSuite = similarityMatrix.getTestCases();
```

Algorithm 1: Similarity-based test reduction algorithm

In Algorithm 1, we can observe a repeating structure (`while` command in line 1) that is executed $\frac{n^2-n}{2}$, where n is the number of test cases in the test suite. Furthermore, within each iteration the method `getAllMaxValue` ($O(n^2)$) is used to search the matrix for the highest similarity values. Therefore the complexity analysis of Algorithm 1 has a complexity of $O(\frac{n^2-n}{2} \times n^2)$.

4. Empirical Studies

In order to evaluate the effectiveness of our proposal, we conducted an experimental investigation. For this investigation, the process for experimental studies in software engineering proposed by Wohlin *et al.* [Wohlin et al. 2000] was applied. The following activities make up this process: definition, planning, operation (described below) and analysis and interpretation, presentation and package (described in the next section).

4.1. Definition

The goal of this experiment is to investigate **test suite reduction strategies**, observing **reduced test suite size and failure coverage**. Based on this goal, our general hypothesis is that *"Test Suite Reduction Strategies show a different performance with respect to reduced test suite size and failure coverage"*. Furthermore, the analyzed results consider the point of view of the tester (responsible for the testing process) in the context of Model-Based Testing (MBT).

4.2. Planning

From the definition of the experiment, in the planning phase was defined how the experiment should be conducted. Steps are defined below.

Context Selection: This experiment was conducted in laboratory, i.e., not an industrial environment and hence the experiment was run *off-line*. The inputs of the reduction strategies investigated were automatically generated LTS specifications (*toy problems*). Although these specifications have distinct configurations, this experiment is focused on only some of them. Therefore, it can be characterized as a *specific* context. Furthermore, this experiment did not involve people (subjects), so it can not be characterized as *student vs. professional*.

Variables Selection: The dependent (observed) and independent (controlled and modified) variables characterize the experiment. For this experiment, the dependent variables chosen are the percentage of the number of test cases that the reduced test suite contains (reduced test suite size – **RTSS**) and the total number of failures that are uncovered by the reduced test suite (failure coverage – **FC**). The independent variables are the test requirements (*all-transition-pair coverage*) and the test suite reduction strategies (G, GE, GRE, H and Sim).

Hypothesis Formulation: Based on the goal of this experiment, we can define the following hypotheses. For each metric (RTSS and FC), we define two hypotheses.

1. **RTSS:** *A null hypothesis (H_1^0):* all strategies have the same behaviour in relation to the reduced test suite size; *An alternative hypothesis (H_1^1):* all strategies have a different behaviour in relation to the reduced test suite size.
$$H_1^0 : RTSS_G = RTSS_{GE} = RTSS_{GRE} = RTSS_H = RTSS_{Sim}$$
$$H_1^1 : RTSS_G \neq RTSS_{GE} \neq RTSS_{GRE} \neq RTSS_H \neq RTSS_{Sim}$$
2. **FC:** *A null hypothesis (H_2^0):* all reduction strategies have the same behaviour in relation to *failure coverage*; *An alternative hypothesis (H_2^1):* all reduction strategies have a different behaviour in relation to *failure coverage*.
$$H_2^0 : FC_G = FC_{GE} = FC_{GRE} = FC_H = FC_{Sim}$$
$$H_2^1 : FC_G \neq FC_{GE} \neq FC_{GRE} \neq FC_H \neq FC_{Sim}$$

Instrumentation: According to Wohlin *et al.* [Wohlin et al. 2000], the instrumentation for this experiment can be characterized by three types of instruments:

1. **Objects:** 12 automatically generated LTS specifications using an adaptation of the LTS generator proposed by Cartaxo [Cartaxo 2011]. These specifications have different configurations, such as: the number for the depth of the LTS, forks, transitions of forks, joins and transitions of joins, and the number of paths with loop (path that goes back to any state already visited in this path [Bondy and Murty 1982]), as presented in Table 1. The failures for each specification model were also automatically defined considering the percentage of 10% of the size of generated test suites. More specifically, 10% of the test cases were randomly chosen and marked as the ones that will produce a failure. That value of percentage of failure (10%) is based on average of number of failures that have been detected in the scope of other studies conducted in our research group. It is important to remark that failures of test cases were considered instead of faults since the latter exist only at code level and cannot be precisely represented at model level.
2. **Guidelines:** no guideline was used, since the strategies do not require people (subjects) to configure them;

3. **Measurements:** LTS generator was used to generate the specifications, and LTS-BT tool was used to support the experiments and collect the data [Cartaxo et al. 2008].

Table 1. The configurations of the specifications used in this experiment

Specification	**01**	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**	**10**	**11**	**12**
Depth	14	14	13	15	17	16	14	16	13	18	15	15
Paths with loop	1	1	1	2	1	1	0	0	0	0	0	0
Forks	23	21	15	19	16	22	22	23	17	23	16	17
Transitions of forks	75	69	47	67	56	60	62	72	51	66	47	45
Joins	21	21	13	14	15	14	14	21	14	14	9	10
Transitions of joins	42	42	27	28	30	28	29	42	28	28	19	20

Experimental Design: As mentioned before, 12 automatically generated specifications were used. For each specification, there is one experimental study of one factor (reduction strategy) with more than two treatments. Therefore, this experiment was structured in two experimental designs, i.e., one experimental design for each metric observed (RTSS and FC). Considering the suggestions in statistical literature for conducting experiments, the chosen confidence level is 95% (significance level is $\alpha = 0.05$) [Jain 1991]. However, aiming to obtain conclusions with statistical significance, the minimum sample size must be calculated. Thus, we executed the five reduction strategies 40 times to calculate the number of replications required (n) according to Jain [Jain 1991] for each metric in each one of the 12 different specifications. The number of necessary replications for the experiment is the highest value defined among the metrics (RTSS and FC) of all specifications defined by heuristic H for specification 12, observing the RTSS metric, as summarized in Table 2. In this case, a total of **1000** replications for each strategy in each specification is needed.

Table 2. Mean, standard deviation and the highest number of necessary replications for each metric and each specification

Metric	**RTSS**	**FC**
Strategy	H	H
Specification	12	01
Mean ($\overline{x}$)	2.155	33.846
Standard Deviation (s)	1.690	10.989
Necessary Replications (n)	**945.62**	162.00

Validity Evaluation: For the results to be validated, it is important to consider the potential threats that may negatively influence on the results. In this sense, Cook and Campbell [Cook and Campbell 1979] suggest that the threats can be identified according to the type of validation of results and defined in a list of four types: *conclusion, internal, construct and external.*

- **Conclusion validity:** a threat to conclusion validity is related to the statistical tests to be used. In order to maintain the statistical significance of the data, the number of executions of each experimental study is eventually higher than the amount defined in the sample size among all the strategies. All the analysis consider the confidence level of 95%, according to suggestions for conducting experiments in statistical literature [Jain 1991]. Furthermore, choices of the statistical tests were performed after the achievement of normality tests. Thus, this ensures that we have a good conclusion validity;
- **Internal validity:** the control of the experiment is the main threat to internal validity. Whereas the execution of the reduction strategies is automatic, during the implementation and execution of the algorithms, the control is added, so that the execution environment is not influenced by other processes, programs, or the machine on which

the experiment was running. Aside from that, in this experiment there are not people involved, and the same inputs (LTS specifications) are applied for all the strategies. Thus, the internal validity is not considered critical;

- **Construct validity:** concerning the construct validity, a threat is related to the experiment setting. For the experimenter not to influence the measures, and to maintain the construct validity, our results rely on input specifications that have a given set of failures. The sets of failures were randomly generated and the number of failures was defined according to the percentage of the test cases of the test suite. Furthermore, the adaptation of the LTS generator proposed by Cartaxo [Cartaxo 2011] is another threat to validity. Our strategy was implemented according to algorithms described in Section 3. Another threat to validity is the implementation of the metrics (RTSS and FC). These metrics were implemented according to concepts proposed in literature;
- **External validity:** the objects used in this experiment are the main threat to external validity. To maintain the external validity, these objects were automatically generated, not representing a real behavior. On the other hand, considering several specifications (randomly generated) we can obtain an overview of the execution of the strategies on a number of different applications.

4.3. Operation

To generate the different specifications automatically, we adapted the implementation of the LTS generator proposed by Cartaxo [Cartaxo 2011]. The class of the experiment was implemented in the Java programming language[1]. Following this, the LTS-BT tool was used to generate test cases as well as to execute all strategies investigated in the experiment. Furthermore, for this experiment each specification was performed the maximum number of times defined among the metrics, by using a machine with Intel Core(TM) i5 3.10 GHz, 8GB RAM running GNU Linux.

5. Experiment Results and Analysis

This section presents and discusses the results obtained from the empirical study defined in Section 4. For this, we used descriptive statistics to numerically and graphically present the data set.

The first step to analyse the data was to check if they have a normal distribution for all specifications, considering RTSS and FC. For this, we applied the Anderson-Darling normality test, using the Minitab tool[2], considering the confidence level at 95% (significance level is $\alpha = 0.05$) [Jain 1991]. These normality tests were applied, and the ρ-*values* were smaller than the significance value ($\alpha = 0.05$) in all cases[3]. Thus, non-parametric tests should be applied. Since each experimental design has a unique factor with more than two treatments, the non-parametric Kruskal-Wallis test was applied to check the null hypothesis. This test is used to determine if there are "*significant*" differences among the population medians.

Considering RTSS, ρ-*values* $= 1.000$ were obtained by executing the Kruskal-Wallis test for the specifications 5, 6, 10 and 11. The same happened to FC of the specifications 8, 10 and 12. Thus, null hypotheses cannot be rejected in those cases. In other words, for these specifications and metrics, the strategies have the same behaviour with

[1]http://www.sun.com/java/

[2]http://www.minitab.com/

[3]For lack of space, all data and graphs are available at http://sites.google.com/site/similaritysast2013/

95% confidence level, as shown in Table 3. For the other cases, the ρ*-values* obtained were smaller than the significance level ($\alpha = 0.05$). Thus, all null hypotheses can be rejected. In other words, with 95% confidence level the strategies can be considered different, considering RTSS and FC.

To clarify the differences, the confidence intervals were plotted for the all specifications used, considering RTSS and FC. For the pairs of strategies that presented overlap in the confidence intervals, the Mann-Whitney test was applied. If the $\rho - value < \alpha$ (0.05) for Mann-Whitney tests, then the null hypothesis can be rejected in favor of the alternative hypothesis. For the other cases, such as $\rho - values > \alpha$ (0.05), the null hypothesis cannot be rejected, and we concluded that the strategies have similar behaviour. From the confidence intervals and Mann-Whitney tests for the strategies with overlap, we could observe the performance of the strategies, as presented in Table 3.

Table 3. Ordination of effectiveness of the reduction strategies

Specification	RTSS	FC
01	$G = GE = GRE > H > Sim$	$Sim > GRE > G = GE = H$
02	$G = GE = GRE = Sim > H$	$H > G = GE = GRE > Sim$
03	$G = GE = GRE = Sim > H$	$Sim > GRE > G = GE = H$
04	$G = GE = GRE > H > Sim$	$Sim > GE = GRE > G > H$
05	$G = GE = GRE = H = Sim$	$GRE > Sim > GE > G = H$
06	$G = GE = GRE = H = Sim$	$G = GE = H > Sim > GRE$
07	$G = GE = GRE > Sim > H$	$Sim > G = GE = GRE = H$
08	$G = GE = GRE > H > Sim$	$G = GE = GRE = H = Sim$
09	$G = GE = GRE = Sim > H$	$Sim > G = GE = GRE = H$
10	$G = GE = GRE = H = Sim$	$G = GE = GRE = H = Sim$
11	$G = GE = GRE = H = Sim$	$Sim > G = GE = GRE > H$
12	$G = GE = GRE = Sim > H$	$G = GE = GRE = H = Sim$
AVERAGE	$G = GE = GRE >$ **Sim** $> H$	**Sim** $> G = GE > GRE > H$

As the context of the study is specific, we cannot generalize the results. However, we can observe the position of each strategy for all specifications and calculate the average position of each strategy. Our strategy in the ordination of effectiveness average is **boldfaced**. On one hand, by analysing the RTSS metric, the best strategies are the heuristics G, GE and GRE, which presented a similar behaviour, followed by **Sim**. On the other hand, by analysing the FC metric, our proposed **Sim** is the best strategy. Overall, as the differences on RTSS are not significant, **Sim** can be considered the most effective technique.

6. Related Work

In literature, different studies have been conducted to propose a reduced test suite from the original suite that covers a given set of test requirements. Several new strategies of test suite reduction have been reported. Most strategies have been proposed for code based suites, but the results cannot be generalized and sometimes they are divergent. A number of these works are based on classical greedy algorithm, such as [Tallam and Gupta 2005, Parsa and Khalilian 2009, Xu et al. 2012]. Other works are focused in experimental studies that aim to compare previous strategies proposed in literature, such as [Chen and Lau 1998b, Zhong et al. 2008, Rothermel et al. 2002]. In the MBT context, few test suite reduction strategies have been proposed. For instance, Heimdahl and George [Heimdahl and George 2004] investigated the use several coverage criteria in an experiment, such as: transition coverage, decision coverage, MC/DC, MC/DC usage, etc. to significantly reduce the sizes of the test suites generated. To reduce the test suite, the algorithm randomly chooses the test cases. If this test case improves the coverage criterion, then it is added to the reduced set. On the other hand, the reduced test suite has a decrease in fault detection effectiveness. In another study, Cartaxo [Cartaxo 2011] presents a new approach to reduce test suites based on dissimilarity in the MBT context. When compared to the strategy proposed in this paper, dissimilarity also includes the essentialness strategy and selection order starts from the less similar test cases. From experimental results, dissimilarity has shown a low percentage of reduction when compared to the heuristics that were considered in this paper, even though the failure coverage presents better results than the heuristics (as well as presented here).

7. Concluding Remarks

In this paper, we present a new strategy to reduce test suites. The proposed strategy is based on similarity among test cases. The key idea in reducing the test suite is to remove the most similar test cases according to a determined set of test requirements. So that the reduced test suites have the less similar test cases, aiming to have a better coverage of both test requirements and failures. On the other hand, the heuristics are always based by the best local choices aiming maximize coverage of the test requirements. The experimental results show that the reduced test suite sizes obtained by applying our strategy is in average greater than that one obtained by applying the heuristics. Furthermore, by analysing the results obtained for each specification in separate, we can see that the difference is not so large (in average 0.326%) and, in 50% of the cases our strategy presents the same behavior as G, GE and GRE. Considering the failure coverage, it can be observed that in average our strategy presents an improvement on failure detection effectiveness compared to other heuristics. Sim can reveal more 6.96% (in average) than the others and also presents a lower standard deviation. Our results suggested that our proposed strategy is promising. Executing more case studies and experiments using other configurations as input of the LTS generator is part of our future work as well as evaluate the strategy using others metrics.

Acknowledgments. This work was supported by CNPq grants 484643/2011-8 and 560014/2010-4. Also, this work was partially supported by the National Institute of Science and Technology for Software Engineering[4], funded by CNPq/Brasil, grant 573964/2008-4. First author is supported by CAPES and the second one is supported by CNPq-Brasil.

References

Bertolino, A., Cartaxo, E., Machado, P., Marchetti, E., and ao Ouriques, J. (2010). Test suite reduction in good order: Comparing heuristics from a new viewpoint. In *Proceedings of the 22nd IFIP International Conference on Testing Software and Systems: Short Papers*, pages 13–18. CRIM.

Bondy, J. and Murty, U. (1982). *Graph Theory with Applications*. North-Holland, NY, USA, fifth edition.

Cartaxo, E. G. (2011). *Estratégias para Controlar o Tamanho da Suíte de Teste Gerada a partir de Abordagens MBT*. PhD thesis, Universidade Federal de Campina Grande, Campina Grande, Paraíba.

Cartaxo, E. G., Andrade, W. L., Neto, F. G. O., and Machado, P. D. L. (2008). LTS-BT: a tool to generate and select functional test cases for embedded systems. In *Proceedings of the 2008 ACM symposium on Applied computing*, SAC'08, pages 1540–1544, New York, NY, USA. ACM.

[4]www.ines.org.br

Cartaxo, E. G., Machado, P. D. L., and Neto, F. G. O. (2011). On the use of a similarity function for test case selection in the context of model-based testing. *STVR Journal of Software Testing, Verification, and Reliability*.

Chen, T. Y. and Lau, M. F. (1998a). A new heuristic for test suite reduction. *Information & Software Technology*, 40(5-6):347–354.

Chen, T. Y. and Lau, M. F. (1998b). A simulation study on some heuristics for test suite reduction. *Information & Software Technology*, 40(13):777–787.

Cook, T. D. and Campbell, D. T. (1979). *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Houghton Mifflin.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press, Cambridge, MA.

Fraser, G. and Wotawa, F. (2007). Redundancy based test-suite reduction. In *Proceedings of the 10th international conference on Fundamental approaches to software engineering*, FASE'07, pages 291–305, Berlin, Heidelberg. Springer-Verlag.

Harrold, M. J., Gupta, R., and Soffa, M. L. (1993). A methodology for controlling the size of a test suite. *ACM Trans. Softw. Eng. Methodol.*, 2(3):270–285.

Heimdahl, M. and George, D. (2004). Test-suite reduction for model based tests: effects on test quality and implications for testing. In *Automated Software Engineering, 2004. Proceedings. 19th International Conference on*, pages 176–185.

Jain, R. (1991). *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley.

Parsa, S. and Khalilian, A. (2009). A bi-objective model inspired greedy algorithm for test suite minimization. In *Proceedings of the 1st International Conference on Future Generation Information Technology*, FGIT '09, pages 208–215, Berlin, Heidelberg. Springer-Verlag.

Rothermel, G., Harrold, M. J., Ronne, J. V., and Hong, C. (2002). Empirical studies of test-suite reduction. *Journal of Software Testing, Verification, and Reliability*, 12:219–249.

Tallam, S. and Gupta, N. (2005). A concept analysis inspired greedy algorithm for test suite minimization. *SIGSOFT Softw. Eng. Notes*, 31(1):35–42.

Utting, M. and Legeard, B. (2007). *Practical Model-Based Testing - A Tools Approach*. Morgan Kaufmann.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslën, A. (2000). *Experimentation in software engineering: an introduction*, volume 15. Kluwer Academic Publishers.

Xu, S., Miao, H., and Gao, H. (2012). Test suite reduction using weighted set covering techniques. In *Software Engineering, Artificial Intelligence, Networking and Parallel Distributed Computing (SNPD), 2012 13th ACIS International Conference on*, pages 307–312.

Zhong, H., Zhang, L., and Mei, H. (2008). An experimental study of four typical test suite reduction techniques. *Information and Software Technology*, 50(6):534–546.