2017 IEEE/ACM 39th International Conference on Software Engineering

Adaptive Coverage and Operational Profile-based Testing for Reliability Improvement

Antonia Bertolino, Breno Miranda
ISTI - CNR
Via G. Moruzzi 1, 56124, Pisa, Italy
{antonia.bertolino, breno.miranda}@isti.cnr.it

Roberto Pietrantuono, Stefano Russo
Università degli Studi di Napoli Federico II
Via Claudio 21, 80125, Napoli, Italy
{roberto.pietrantuono, stefano.russo}@unina.it

***Abstract*—We introduce *covrel*, an adaptive software testing approach based on the combined use of *operational profile* and *coverage spectrum*, with the ultimate goal of improving the delivered reliability of the program under test. *Operational profile-based testing* is a black-box technique that selects test cases having the largest impact on failure probability in operation; as such, it is considered well suited when reliability is a major concern. *Program spectrum* is a characterization of a program's behavior in terms of the code entities (e.g., branches, statements, functions) that are covered as the program executes. The driving idea of *covrel* is to complement operational profile information with white-box coverage measures based on count spectra, so as to dynamically select the most effective test cases for reliability improvement. In particular, we bias operational profile-based test selection towards those entities covered less frequently. We assess the approach by experiments with 18 versions from 4 subjects commonly used in software testing research, comparing results with traditional operational and coverage testing. Results show that exploiting operational and coverage data in a combined adaptive way actually pays in terms of reliability improvement, with *covrel* overcoming conventional operational testing in more than 80% of the cases.**

***Keywords*-Testing; Reliability; Operational profile; Program count spectrum; Operational coverage; Test case selection.**

I. Introduction

Testing based on the operational profile, i.e., *"as if it were in the field"* [1], is a fundamental technique in software reliability engineering practice [2]. The aim is to select those test inputs that can expose failures having a higher occurrence probability, based on the known or estimated operational profile, and hence contributing more to program unreliability. Operational profile-based testing is widely used for both reliability *assessment* [3] [4] and reliability *improvement* (e.g., within the SRET technique [5]). In this paper we address the latter goal.

Several studies document the effectiveness of operational profile-based testing for reliability improvement [6] [7]. However, as for any test selection technique, it may suffer from the saturation effect ([2], Chapter 13), due to which its continued application will eventually lose efficacy. In other words, to further improve reliability beyond a certain point, testing should try to expose also those failures having low probability of occurrence based on the operational profile [8].

In [9] the authors provide a formal reasoning model to support the intuition that since different techniques are effective at discovering different faults, it is better to use a combination of approaches rather than relying on one technique alone. This is discussed also in [10], arguing that the combination of techniques should aim at exposing high-occurrence failure regions, but also as many failure regions as possible. Here we apply the intuition of combining techniques by mixing operational profile-based and coverage testing into a novel hybrid technique, named *covrel*.

The approach is inspired by the notion of *operational coverage* introduced in [11], which clusters entities into different "importance" groups according to their count spectra, and assigns them different weights while computing coverage. A spectrum provides a signature of a program's behaviour by tracking the coverage of entities (e.g., statements, branches, or even whole paths) during execution [12]. Traditionally, coverage-based testing approaches consider *hit spectra*, i.e., they only measure which entities have been executed (at least once). *Count spectra* provide richer information, by also measuring the number of times each entity is executed, and are typically used in program optimization [13].

In [11] operational coverage was studied as an adequacy criterion for operational profile-based testing. Here we exploit operational coverage to support test selection, precisely we propose that among several test cases that would have a same estimated usage probability in the operational profile, we select those that exercise entities yielding so far a low count spectrum. Moreover, the approach is adaptive, in that testing proceeds in iterations (as previously done in [8]), and at each iteration both the profile-based allocation of number of tests to partitions and operational coverage measures are dynamically updated.

As a result, *covrel* selects the test cases according to the (black-box) operational profile (i.e., the input domain partitions more often invoked in operation are more exercised) *and*, among the inputs within a same partition, *covrel* selects those that exercise the least covered entities. To the best of our knowledge, this is the first proposal encompassing the usage of coverage spectra to guide test selection in the framework of a reliability-oriented testing technique.

The type of software systems targeted in "*covrel*ing" are those for which reliability is a driving concern of testing activities. Examples are large-scale mission-critical systems, for which the time (and cost) of executing individual tests is high (due to configuration, set up, execution and shutdown).

1558-1225/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE.2017.56

For instance, a study conducted in an industrial context in [14] has shown that the fault detection rate for complex components may decrease significantly over time, giving rise to the need to improve the trend of testing efficacy (i.e., of the level of defectiveness exposed by testers). For such industrial applications it is particularly important that when operational testing enters the saturation area, the marginal cost of executing additional tests pays in terms of reliability improvement. In *covrel* we pay the potential improvement in reliability with the cost of measuring coverage.

We experimented *covrel* on a number of subjects from the public Software-artifact Infrastructure Repository (SIR), widely used in software testing research [15]. We compare *covrel* with traditional operational and coverage testing on the basis of several metrics. As we expected, the results show that *covrel* generally outperforms each of them used in isolation when targeting reliability improvement.

In summary, this paper provides the following contributions:

- We develop the *covrel* approach to testing, which combines operational profile and coverage spectrum information for improvement of delivered reliability;
- We evaluate *covrel* with several subjects from the well-known SIR repository, and compare it empirically against traditional operational and coverage testing;
- We provide a prototypal implementation of *covrel* for repeatability and independent verifiability.

The rest of the paper is structured as follows. Section II provides background information and overviews related work. Section III presents the proposed *covrel* technique. Section IV describes the experiments. Section V discusses the results and their statistical significance, as well as the threats to validity; it also describes the artifacts we produced for implementing *covrel*, which we make available with the aim to support verifiability. Section VI contains concluding remarks and future work.

II. Related Work

The literature of software reliability and operational profile-based testing is huge. The underlying idea is to test a program by selecting an input t according to a probability p_t derived from the operational profile distribution. An operational profile is a quantitative characterization of how a system will be used, and is usually built by assigning probability values to inputs representing the probability that each will occur in operation (e.g., by historical data, by design-level information, by expert judgment). We refer to the highly referenced and still relevant Lyu's handbook [2] for a comprehensive overview of the topic, and for the rest of this section we focus on related works that from different perspectives explore the relation between reliability and code coverage.

Many empirical studies, e.g., [16] [17] among the most recent ones, assess the effectiveness of coverage-based testing. However only a few of them measure test effectiveness in terms of delivered reliability [6], as we do here. Among the earliest studies, Del Frate and coauthors [18] found a correlation between increase (decrease) in reliability and increase (decrease) in at least one code coverage measure. In a later work Frankl and Deng [19] performed a case study comparing various approaches and showed that as coverage increases, the probability to achieve high reliability targets increases as well. However, they show also that the probability to reach very high reliability values would require extremely large test sets, and it is doubtful whether the improvement is worth the cost. This is what motivates our work: *covrel* explores the usage of coverage *in combination* with traditional operational testing for reliability improvement.

Several authors have proposed to integrate test coverage information into models used to evaluate reliability as faults are found and removed (a.k.a. Software Reliability Growth Models or SRGMs). A recent short compendium of such coverage-integrated SRGMs is given by Alrmuny [20]. Although the basic motivation is the same, i.e., that reliability improvement can be impacted by observed coverage measures, the usage that we make of such measures is different. In SRGMs coverage information is used to better tune reliability estimation, as in [21]; in *covrel* we use coverage information for driving test selection, building on the concept that coverage measures can provide guidance in identifying what parts of a program should be exercised when augmenting a test suite [22] [23].

Program spectra [12] are used extensively in software analysis and testing [24]. Beyond their original application in program optimization [13], in [25] Reps et al. introduced the idea that differences between path spectra can be used for identifying changes in program behaviour. Following [25], code profiling information has been actively used to analyze the executions of different versions of programs, e.g. in regression testing [26], or to compare traces of failed and successful runs in fault diagnosis [27] [28]. Differently from these approaches, we do not use spectra to differentiate behaviors, but are interested in identifying which entities have been less exercised.

III. Approach

A. Assumptions

Let us denote the input domain of the program under test as D. Suppose a tester has a budget of T test cases to select from a test suite. We make the following assumptions:

- The input domain D can be decomposed into M subdomains (hereafter also called partitions) $D_1, \ldots, D_m$. These are divided according to some partitioning criterion (e.g., functional or structural), usually depending on the information available to test designers and on testing objectives.
- The operational profile can be described as a probability distribution over partitions D_i. We denote with p_i the probability that an input is selected from D_i in operation, and $\sum_{i=1}^{M} p_i = 1$. Inputs within a partition have the same probability of runtime occurrence.
- A test case leads to failure or success; we are able to determine when it is successful or not (perfect oracle).
- Test case runs are independent, i.e., all the non-executed test cases are admissible each time. The execution of a

test case is not constrained by the execution of some other test case before. This affects the way in which a "test case" is defined, since when test cases are sequences of tasks, they can be grouped together in a single test case, so that at the end of the test case the system goes back to the initial state [29].
- The output of a test case is independent of the history of testing; in other words, a failing test case is always such, independently of the previously run test cases.
- A test suite from which test cases can be selected exists, and coverage information of such test cases is available or can be obtained.

B. *The* covrel *strategy: overview*

The objective of *covrel* is to improve delivered reliability efficiently by means of a focused test case selection strategy. The strategy proceeds adaptively in subsequent iterations, and the underlying idea is to dynamically gain knowledge along testing execution about *i)* which regions of the input space, and then *ii)* which specific test cases within each region, are expected to contribute more to reliability improvement at the next iteration.

In line with [6], reliability is defined as $R = 1 - \sum_{t \in F} p_t$, where F is the set of inputs leading to failure (i.e., the failure points) and p_t is the expected probability of occurrence in operation of input t. Thus, the reliability "contribution" of a set of test cases depends on the number and size of the detected failure regions[1] (i.e., how many failure points are corrected by the removal of the detected faults) and on their expected probability of execution in operation. It may well happen that eliminating few but frequent failure points is better than eliminating many infrequent ones – so a balance between these two contrasting objectives is needed. As detected failure points are removed during the testing and debugging process, the status of the input domain changes; this requires adaptivity, if we want to look always for the best balance along the entire process.

Covrel is designed to iteratively search for those test cases with potentially the highest contribution to (un)reliability, by combining the ability of finding still undetected faults with the ability of finding high-occurrence ones. It uses *learning* and *adaptation* to dynamically characterize the input domain partitions and select test cases from them. The strategy is sketched in Figure 1. The iterative process foresees two main phases: the **allocation** of test cases to partitions D_i, and the **selection** of test cases within each partition. It is conceived to exploit the results of test execution per partition (namely, number of exposed failures and current coverage) at each iteration, in order to drive test allocation and selection at successive iterations. The steps are described in detail in the following.

[1]A failure region is the set of failure points eliminated by a program change [30].

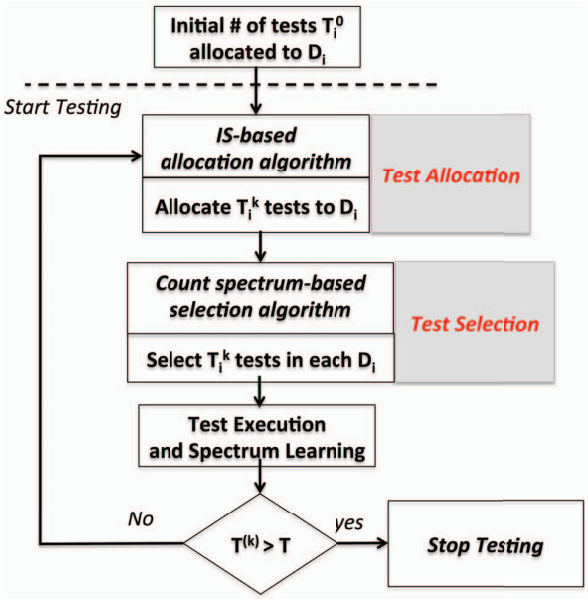


Fig. 1: The *covrel* strategy

C. *Phase 1: Allocation of test cases to partitions*

Covrel's adaptiveness aims at periodically re-allocating test cases depending on where more tests are actually needed. This is accomplished by means of an algorithm based on the *Importance Sampling* (IS) method [31], which is an inference method to approximate the *true* distribution of a variable of interest that we also used in previous work [8]. In *covrel*, the *true* unknown distribution of interest is the best number of test cases for each partition that would maximize the delivered reliability. The algorithm represents the beliefs (i.e., hypotheses) about this distribution by means of sets of "samples". Each sample is associated with a probability that the belief is true: at each iteration, these probabilities are updated by examining some new samples of that hypothesis, and a larger number of samples are drawn from hypotheses with a larger probability. The goal is to converge, in few iterations, to the "true" best distribution of test cases.

To establish how the probability of each hypothesis is updated based on new collected samples, an update rule is defined. At a given iteration k, the IS-based algorithm exploits information from previous iterations about the observed failing tests. The failure rate (denoted with φ_i), defined as number of failing tests over number of executed ones, is used to smoothly direct testing toward the partitions with a high expected (un)reliability contribution in the next iteration. Specifically, let us denote with $\pi^{(k)}$ the probability vector representing, for each partition i and at iteration k, the likelihoods that *testing from that partition contributes to improve delivered reliability*. Denoting with $\theta_i = p_i \varphi_i$ the weighted failure rate (normalized

so that $\sum_i \theta_i = 1$), the update rule for the values $\pi_i^{(k)}$ forming the probability vector is defined as:

$$\pi_i^{(k)} = \gamma \pi_i^{(k-1)} + (1-\gamma) \cdot (1 - \theta_i^{(k-1)}). \quad (1)$$

Such assignment tends to explore the input domain by progressively moving tests to partitions where unreliability contribution was still small; this allows detecting hard-to-find faults after easier ones have been found. The smoothness of adaptiveness is determined by the parameter $\gamma \in [0,1]$, regulating how the algorithm considers past iterations' results with respect to current ones[2]. Note that the $\pi_i^{(k)}$ values are normalized, since they are probabilities ($\pi_i^{(k)} = (\pi_i^{(k)})/(\sum_{i \in D} \pi_i^{(k)})$). Starting from $\pi_i^{(k)}$, $T_i^{(k+1)}$ tests are allocated to partition D_i at iteration k by a simple procedure to assign, proportionally, more tests to domains with higher p_i values [8], so as: $T_i^{(k+1)} \approx T^{(k+1)} \pi_i^{(k)}$.

To determine the best $T^{(k+1)}$ at each iteration, we consider an *adaptive* implementation of importance sampling [32]. Based on a desired error and confidence, this variant tends to progressively reduce the number of required samples as more information becomes available, using the formula:

$$T^{(k+1)} = \frac{1}{2\xi} \chi^2_{\rho-1,1-\delta} \approx \frac{\rho-1}{2\xi} \{1 - \frac{2}{9(\rho-1)} + \sqrt{\frac{2}{9(\rho-1)}} z_{1-\delta}\}^3 \quad (2)$$

where: ξ is the error we want to tolerate between the sampling-based estimate and the true distribution; $1-\delta$ is the desired confidence in this approximation; ρ is the number of partitions from which at least one test case has been drawn in the k-th iteration; $z_{1-\delta}$ is the normal distribution evaluated with significance level δ.

The algorithm is triggered by an initial static allocation of $T^{(0)}$ tests at iteration 0. Several alternatives can be chosen for such initial allocation depending on the initial knowledge about failure likelihood of partitions (e.g., via expert judgment about partition criticality). We assume that no information is available, and allocate tests proportionally to the expected usage of the partition p_i – so giving more tests to partitions whose inputs are expected to be more exercised. The number of tests $T^{(0)}$ must be a small subset of T, as it is just required to trigger the allocation algorithm [33][3].

Summarizing, the output of this phase is the computation of the most proper number of test cases $T^{(k)}$ to run and their allocation to partition D_i, namely $T_i^{(k)}$. In the following we detail how these tests are selected within partitions.

D. Phase 2: Selection of test cases within a partition

While the previous "allocation step" suggests the number $T_i^{(k)}$ of test cases to execute for each partition, the "selection step" cares about picking these $T_i^{(k)}$ test cases among the ones not yet executed (without-replacement selection) in an effective way.

[2]We set γ at 0.5 in our experiments.

[3]In general, the bigger $T^{(0)}$ is, the better the initial learning could be, but the later the adaptation will start. In our experiment, we opted for $T^{(0)}$ equal to the number of partitions.

As we assume that no initial knowledge is available about the failure likelihood of partitions, at the first iteration of *covrel* the test cases are selected following a traditional operational profile based testing, i.e., they are just randomly selected according to occurrence probability of each partition D_i.

However, while the test cases are executed, their traces are tracked, as in any white-box testing strategy. At the end of each iteration, the learning phase takes place: the *covrel* approach computes the cumulative count spectrum achieved during the execution of the selected test cases and uses it to drive the selection strategy for the next iteration.

More precisely, the cumulative count spectrum is used to identify the frequency with which entities have been exercised. For the experiments we report in Section IV, we classified the entities into three different importance groups: *high*, *medium*, and *low*. To obtain such groups, we ordered the entities available in the count spectrum according to their frequency of usage and assigned the top 1/3 entities to the *high* frequency group; the second 1/3 entities to the *medium* frequency group; and the last 1/3 entities to the *low* frequency group.

Next, our approach evaluates each test case that belongs to the target partition D_i and ranks them according to *how* they cover the program entities. For computing the test case rank, we assign weights to each importance group. As we are interested in selecting test cases that cover entities so far rarely exercised, we assign the weights for the importance groups in such a way that the medium group is one order of magnitude more important than the high, and the low group, on its turn, is one order of magnitude more important than the medium group (e.g., $W_{high} = 10^{-1}$, $W_{medium} = 10^0$, and $W_{low} = 10^1$).

For a given partition D_i, the test cases with the highest ranks are the ones selected to compose the $T_i^{(k)}$ set. In the case of a tie, i.e., multiple test cases would achieve the same rank, our heuristic randomly selects one test case among the tied ones. The allocation and selection steps are repeated in *covrel* until the number of available test cases T has been run, as depicted in Figure 1.

IV. Experiments

We performed a controlled experiment to assess the effectiveness of *covrel* in reliability improvement. The experiment aimed at answering the following research question: *Is covrel more effective at reliability improvement than traditional operational profile-based testing?* For our setting, more effective means being able to deliver a given reliability value with fewer test cases.

We consider various testing scenarios under different operational profiles; each scenario is determined by the subject program under test, the coverage criterion adopted by the *covrel* test selection algorithm (e.g., function, branch or statement coverage), and the selection of seeded faults. These factors are described in the remainder of this section.

A. Study Subjects

We evaluate *covrel* on a total of 18 versions from four subjects taken from SIR: Grep, Gzip, Sed, and Flex. Grep

is a command-line utility for searching lines matching a given regular expression in the provided file(s); Gzip is an application used for file compression and decompression; Flex is used for generating scanners that are able to recognize lexical patterns in text; and Sed is a stream editor that performs text transformations on an input stream. Three of these programs (Grep, Gzip, and Flex) are available from SIR with 5 variant versions containing seeded faults, whereas Sed contains 7 variant versions. All of them have test suites derived according to the category-partition method available via *tsl* (test specification language) files.

Because our study required us to measure the reliability delivered by the evaluated approaches at different stages of the test execution, we excluded the versions where the test suite available from SIR could not identify any of the seeded faults available for that particular version. This happened for Grep v5, Gzip v3, and Sed v1. Besides that, we also excluded version 5 of Flex due to its anomalous characteristics: 3 faults could be revealed by more than 80% of the test cases, while 2 other faults could be revealed by ≈99% of tests; the vast majority of the test cases available in the test suite would be able to reveal 100% of the seeded faults – so all the faults would be detected after few tests.

Additional details about our study subjects are displayed in Table I. Column "LoC" shows the lines of code[4] of each variant version. The column "Detectable faults" contains the number of faults, from the set of seeded faults, that could be detected by the SIR test suite, whereas the column "Hard faults" refers to the subset of detectable faults that could be revealed by less than 50% of the available test cases.

TABLE I: Study subjects considered in our investigations

Subject	LoC	Test Suite	Seeded faults	Detectable faults	"Hard" faults
Grep v1	9463	809	18	5	4
Grep v2	9987	809	8	4	4
Grep v3	10124	809	18	8	5
Grep v4	10143	809	12	3	3
Gzip v1	4594	214	16	7	6
Gzip v2	5083	214	7	3	1
Gzip v4	5233	214	12	3	3
Gzip v5	5745	214	14	5	4
Sed v2	9867	360	5	5	3
Sed v3	7146	360	6	6	5
Sed v4	7086	363	4	1	1
Sed v5	13398	370	4	4	4
Sed v6	13413	370	6	6	6
Sed v7	14456	370	4	4	4
Flex v1	9558	567	19	16	8
Flex v2	10274	670	20	13	9
Flex v3	10296	670	17	9	9
Flex v4	11447	670	16	11	8
Total:	167313	8862	206	113	87

B. Study Settings

Besides the study subjects acquired from SIR, our experiments required the following additional artifacts.

[4]Collected using the CLOC utility (http://cloc.sourceforge.net/).

Test Suite. We used the test suites from SIR that are available along with each one of the subjects investigated in our study. The number of test cases available in each test suite is provided in Table I.

Partitions. We assign each test case in the test suite to a different partition based on the functionality exercised by the test input. To this aim, we inspected the *tsl* (test specification language) file made available with the subject (which specifies the function units and their input space features in terms of parameters, categories and choices, according to category-partition testing terminology) along with the user manual in order to infer the main functionalities. Each functionality is associated with a partition; we ended up with 4 partitions for Grep, 5 for Gzip, 3 for Sed, and 6 for Flex. We then derived a script to automatically parse the available test cases and allocate each test case to its respective partition based on the functionality it exercises.

Fault Matrix. For each subject and version, we run the available test suite and, with the support of SIR tools, we extracted the fault-matrix, i.e., a mapping that tells us which faults can be revealed by which test cases. Because in our study we want to evaluate the effectiveness of the proposed approach to reveal the most relevant faults, besides the original fault-matrix we derived a second modified one (we refer to it as "hard fault-matrix") by excluding the *easy-to-find* faults that could be revealed by 50% (or more) test cases.

Operational Profile. A *testing configuration* is a combination of the fault matrix and the coverage criteria adopted by *covrel* (i.e., function, branch, and statement). A *testing scenario* is a testing configuration applied to a program-version pair. For each testing scenario, we derived 50 operational profiles. An operational profile is a quantitative characterization of how a system will be used [1]: it can be described as a set of values, p_i, denoting the probability that an input is selected from partition i and such that $\sum_{i=1}^{m} p_i = 1$, with m being the number of partitions [29]. Profiles are built by generating the p_i values according to a uniform distribution in [0,1], then normalizing the values so that they sum up to 1.

C. Compared Techniques

We compare *covrel* against conventional operational testing, where test cases are selected based exclusively on the operational profile estimate [5]. The implemented operational testing technique (hereafter OT) selects a partition D_i from the given test suite at random in accordance with its usage probability $\{p_i\}$. A test within the partition is then selected with equal probability, i.e., by a uniform distribution, similarly to related literature (e.g., [4], [29]).

We examined *covrel* while targeting different coverage criteria (function, branch, and statement). As we want *covrel* to bias the test selection towards test cases covering entities more rarely exercised, we set the weights for the importance groups as: $W_{high} = 10^{-1}$, $W_{medium} = 10^{0}$, and $W_{low} = 10^{1}$.

The importance sampling parameters (see Section III-C) are set as follows: the learning factor is $\gamma = 0.5$; the confidence is $(1 - \delta) = 0.95$; the desired maximum error is $\xi = 0.2$ for

Grep, Sed and Flex; and $\xi = 0.1$ for Gzip, as the lower ratio of test cases over partitions suggests a faster adaptation.

D. Number of runs

Considering the above factors, we have 2 (fault matrices) X 3 (coverage criteria) = 6 testing configurations, each applied to the 18 program-version pairs, getting to 108 different *testing scenarios*. In each testing scenario, we compare the two testing techniques (*covrel* and operational testing) by running 50 testing sessions are performed (one per generated operational profile). The total number of runs thus is: 108 testing scenarios X 50 profiles X 2 compared techniques = 10,800 runs.

E. Evaluation Metrics

To address the research question, we compare the techniques in terms of testing reliability, namely probability of not failing during testing [34][5] *vs* number of test cases.

Specifically, in each testing session we measure the number of test cases required to achieve reliability 1 (i.e., to reveal all the faults that can be detected by that test suite) by the two compared techniques, let us denote them as N_{covrel} and N_{OT}, respectively. Then, as a summary metric, in each given testing scenario we count the number of times $N_{covrel} < N_{OT}$ (i.e., *covrel* achieves reliability 1 with fewer test cases than traditional operational testing) or the vice versa over the 50 repetitions.

Moreover, in order to evaluate how the techniques perform during a testing session, we assess the reliability growth at different checkpoints. Specifically, we observe delivered reliability after the execution of 10%, 50% and 90% of total executed test cases to achieve reliability 1[6]. We count the number of wins (in this case, in terms of achieved reliability) of each technique at every checkpoint.

F. Experiment Procedure

The automated procedure followed for each testing session includes these steps:

1) Generate the operational profile as explained above.
2) Select the test case according to the technique under evaluation (*covrel* and OT) and observe if it exposes a failure or not. The detection of failures is done by considering the fault matrices.
3) If a failure occurs, remove the fault[7].
4) Repeat from step 2 until T test cases are executed.
5) At the end of the session, compute the metrics presented above, useful for evaluation.

[5]In general, the available test cases are not able to detect all the faults, like in the experimented subjects (cf. with Table I), so even when testing reliability is 1 (i.e., all faults detectable by the test suite are detected), the actual operational reliability is likely not 1 because undetected faults could be exposed in operation. Since operational reliability is hard to measure, testing reliability is adopted for comparison purposes.

[6]We take the minimum of N_{covrel} and N_{OT}, and consider the 10%, 50% and 90% of it.

[7]Note that for one failure, the tester could remove more faults (failure regions may not be disjoint); in our case we choose to remove all the faults, i.e., the repetition of the test case must no longer lead to failure

V. Results

In this section we report the results from the experiments performed to answer our research question whether *covrel* is more effective at reliability improvement than OT. We measure effectiveness in reliability improvement in two different manners: respective number of test cases required to reach a same high reliability value, fixed to 1 in the reported study (Section V-A); and respective values of reliability achieved with a same fixed number of test cases (Section V-B).

Although coverage-based testing has a different objective, by discovering faults it can improve reliability. As *covrel* combines operational and coverage information, we would like to compare it not only with OT, but also with coverage-based testing, to assess if our combination outperforms each of the two approaches taken alone in improving reliability. The results of a further evaluation assessing *covrel vs* traditional coverage-based selection are reported in Section V-C.

A. Which approach achieves reliability 1 faster?

In Table II we report the results from the six configurations tested (combinations of coverage strategy and fault matrix, see Section IV-F), counting for each: the number of times (out of 50 different observations) in which *covrel* outperforms OT and wins, in Column W; the number of times *covrel* loses, in Column L; the ties in Column T. At bottom of the same table, two rows report the Mean and Median values for each column.

As evident by the Mean and Median values, our approach is generally more effective in reaching maximum reliability. In all cases the Mean and Median of win counts are much higher than losses, and also higher considering losses and ties summed together.

Looking more in detail, we can observe that if we consider each pair subject&configuration (i.e., what we called a testing scenario) as a distinct experiment, we got 96 valid results (12 pairs among the possible 18*6=108 ones are marked as N/A because for such cases the "hard" fault matrix and the original one were the same – see Section IV-A). Among these, *covrel* wins, i.e. achieves reliability 1 with a lower number of test cases than OT, in 77 cases, i.e., more than 80% of the times.

The above result does not distinguish between clear and tight wins, i.e., regardless whether the result is 50 wins over 0 losses (e.g., as for Grep v1 in Configurations 5 and 6) or 26 wins against 21 losses (as for Flex v1 in Configuration 2), it is anyhow counted as 1. If we consider each repetition on a different profile as a different observation, and consider the sum of wins against sum of losses, then we can observe that *covrel* outperforms OT in all configurations (i.e., summing the results per columns), and for 14 subjects out of 18 (i.e., summing the results per rows). Precisely, *covrel* cumulatively loses with Grep v2, Gzip v2, Sed v7 and Flex v3, and wins for all other subjects.

It is interesting to look at the cases where *covrel* performs worse than OT. In general (with the only exception of Sed v3, for which *covrel* loses only in 1 configuration) when this happens, it happens consistently in all configurations,

TABLE II: Number of wins/losses/ties over 50 repetitions in terms of number of required test cases to achieve reliability=1

Subject	Config. 1 *Function - "hard" FM*			Config. 2 *Branch - "hard" FM*			Config. 3 *Statement - "hard" FM*			Config. 4 *Function - "default" FM*			Config. 5 *Branch - "default" FM*			Config. 6 *Statement - "default" FM*		
	W	*L*	*T*	*W*	*L*	*T*	*W*	*L*	*T*	*W*	*L*	*T*	*W*	*L*	*T*	*W*	*L*	*T*
Grep v1	49	1	0	49	1	0	48	2	0	49	1	0	50	0	0	50	0	0
Grep v2	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1	49	0	10	40	0	9	41	0
Grep v3	47	3	0	29	21	0	32	18	0	40	10	0	27	23	0	37	12	1
Grep v4	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	19	31	0	49	1	0	49	1	0
Gzip v1	48	2	0	42	7	1	46	4	0	35	15	0	34	16	0	33	17	0
Gzip v2	2	35	13	0	33	17	3	33	14	32	2	16	2	34	14	5	32	13
Gzip v4	49	0	1	50	0	0	49	1	0	49	1	0	50	0	0	50	0	0
Gzip v5	44	5	1	45	5	0	42	8	0	45	5	0	42	8	0	46	4	0
Sed v2	38	12	0	40	10	0	46	4	0	44	6	0	34	16	0	36	14	0
Sed v3	14	36	0	38	12	0	45	5	0	33	17	0	43	6	1	46	3	1
Sed v4	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	48	1	1	48	1	1	48	2	0
Sed v5	41	6	3	35	10	5	36	9	5	35	7	8	39	10	1	46	3	1
Sed v6	39	11	0	41	6	3	41	8	1	39	11	0	41	9	0	42	7	1
Sed v7	12	37	1	23	27	0	17	31	2	9	40	1	22	28	0	21	29	0
Flex v1	33	14	3	26	21	3	32	12	6	32	14	4	31	17	2	25	21	4
Flex v2	45	5	0	43	7	0	41	9	0	47	3	0	41	9	0	44	6	0
Flex v3	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	16	34	0	2	48	0	4	46	0
Flex v4	36	14	0	31	19	0	28	22	0	42	8	0	30	20	0	28	21	1
Mean	**35.5**	**12.92**	**1.57**	**35.14**	**12.78**	**2.07**	**36.14**	**11.85**	**2**	**34.16**	**14.16**	**1.66**	**33.05**	**15.88**	**1.05**	**34.38**	**14.38**	**1.22**
Median	40	8.5	0	39	10	0	41	8.5	0	37	9	0	36,5	13	0	39.5	9.5	0.5

hinting at a possible explanation that perhaps there are types of failure regions for which our approach does not perform well. However, further empirical investigation is required before a general conclusion can be drawn.

On the other hand, especially interesting for us is the overall good result over configurations. This result shows that the idea of combining operational profile-based testing with coverage spectra is promising across different coverage criteria, and for both "easy" and "hard" faults.

We further evaluated the above results with statistical analysis. Given a testing scenario, we consider the above number of times a technique wins as our performance measure. We used the Wilcoxon signed-rank test[8] to assess the null hypothesis that the difference between the number of wins in the two compared cases follows a symmetric distribution around zero, namely they are statistically equivalent. Results are displayed in Table IIIa and they clearly show the high confidence on rejecting the null hypothesis as well as the big difference between the overall mean (and median) values of the number of wins.

TABLE III: Hypothesis tests. Text in boldface indicates that the difference is significant at least at 0.05

(a) Number of wins in terms of test cases to achieve reliability 1

	Pairwise Comparison	
	covrel	*OT*
Mean	34.62	13.81
Median	39.00	9.50
p-value	**3.1270e-09**	-

(b) Number of wins in terms of reliability at checkpoints

	Pairwise Comparison		
	10%	*50%*	*90%*
covrel Mean	11.40	17.12	16.26
OT Mean	8.5	11.85	13.77
covrel Median	7	16	15
OT Median	8	9	10
$covrel - OT$ *p-value*	0.2168	**0.0087**	0.4156

B. How do covrel and OT compare in terms of delivered reliability evolution?

In practice, it may not be realistic to assume that testing can continue until reliability achieves 1. So, it may be also interesting to look at the results achieved at intermediate stages of testing. We thus compared the effectiveness of *covrel* against OT at three intermediate checkpoints, namely, after the execution of 10%, 50% and 90% of tests run to get reliability 1, as defined in Section IV-E. At each checkpoint we measured the achieved reliability by either approach and counted which one is yielding a higher value. For the sake of space, we report the Mean and Median results of the number of wins and losses of *covrel* against OT (Table IV).

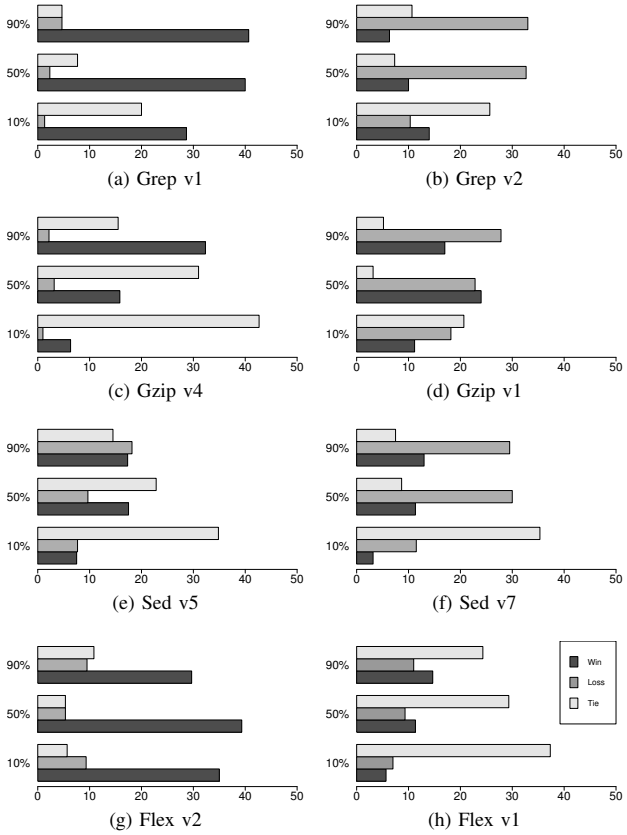
Figure 2 shows a sample of cases, specifically we observed the delivered reliability of all subjects at 90%, and we picked the version in which *covrel* performs best (on the left), and the version in which performs worst (on the right). From this figure we can see that on Flex *covrel* tends to be consistently better than OT, the opposite happens on Sed, and finally on Grep and Gzip the results vary across versions. So, again these results hint at the need to continue experimentation of further subject programs for generalization.

In this experiment we are interested to observe the trend while reliability grows. Our expectation is that *covrel* would yield better results in comparison with OT for higher values of reliability (i.e., when OT starts suffering from the saturation effect). Although results are again promising (the mean values,

[8]We could alternatively apply multiple sign (wins, loss, ties) tests with multiple comparison protection procedure, but we opted for Wilcoxon signed-rank test by treating the number of wins as our performance variable, since it is indeed more powerful [35].

TABLE IV: Mean and Median values of wins/losses of *covrel* in terms of delivered reliability over 50 repetitions at three different checkpoints: 10%, 50% and 90%

Subject	Result	Mean			Median		
		10%	*50%*	*90%*	*10%*	*50%*	*90%*
Grep v1	*wins*	28,67	40,00	40,67	22,5	41,5	39,5
	losses	1,33	2,33	4,67	1	2	3,5
Grep v2	*wins*	14,00	10,00	6,33	14	8	5
	losses	10,33	32,67	33,00	11	35	38
Grep v3	*wins*	7,33	16,83	15,67	7,5	16,5	15,5
	losses	10,83	27,17	28,00	9,5	28,5	27
Grep v4	*wins*	22,67	20,33	16,00	23	23	23
	losses	6,67	1,00	0,33	8	0	0
Gzip v1	*wins*	11,17	24,00	17,00	12	23	17
	losses	18,17	22,83	27,83	20,5	23	27,5
Gzip v2	*wins*	0,67	2,33	5,00	0	0,5	4
	losses	0,67	2,50	3,83	0	0	2
Gzip v4	*wins*	6,33	15,83	32,33	3,5	15,5	33,5
	losses	1,00	3,17	2,17	0	2	1,5
Gzip v5	*wins*	11,50	23,83	19,67	10	22,5	18,5
	losses	7,00	11,50	10,83	7	11	10,5
Sed v2	*wins*	7,83	8,50	4,33	7,5	6	1,5
	losses	21,00	2,67	2,17	19,5	1,5	1
Sed v3	*wins*	18,50	22,67	15,00	19	23,5	16
	losses	10,00	19,00	29,17	9,5	17,5	30
Sed v4	*wins*	0,00	0,00	0,00	0	0	0
	losses	0,00	0,00	0,00	0	0	0
Sed v5	*wins*	7,50	17,50	17,33	7,5	17	15
	losses	7,67	9,67	18,17	8	9,5	20,5
Sed v6	*wins*	3,83	8,17	13,00	4	8	12,5
	losses	10,33	24,00	24,67	11	23,5	24
Sed v7	*wins*	3,17	11,33	13,00	2	12	13
	losses	11,50	30,00	29,50	12	29	30
Flex v1	*wins*	5,67	11,33	14,67	5,5	12,5	16
	losses	7,00	9,33	11,00	7	9,5	10,5
Flex v2	*wins*	35,00	39,33	29,67	35,5	44	32
	losses	9,33	5,33	9,50	10	4,5	7
Flex v3	*wins*	40,00	30,00	19,67	41	28	20
	losses	5,00	2,00	0,67	5	2	1
Flex v4	*wins*	3,17	2,17	1,83	3	1,5	1
	losses	3,17	2,33	1,83	2,5	1,5	1



(a) Grep v1 (b) Grep v2 (c) Gzip v4 (d) Gzip v1 (e) Sed v5 (f) Sed v7 (g) Flex v2 (h) Flex v1

Fig. 2: *Covrel vs* OT at checkpoints

at each checkpoint, of *covrel*'s wins over subjects in a given configuration are bigger than corresponding losses in all the 6 configurations X 3 checkpoints = 18 cases), we did not observe such an increasing trend across checkpoints; on the contrary *covrel* wins more often at the 50% checkpoint. In particular, if we take the mean values of the number of wins at the three checkpoints (reported in the first two rows of Table IIIb), we see that: at the 10% the difference between *covrel* and OT is around 3.5; at the 50% it is about 5.2, and at the 90% the gain reduces to 2.5 (similar considerations stand for medians).

The hypothesis test compares the number of wins of the two approaches at the three checkpoints. Results are displayed in Table IIIb; the test achieved 0.05 p-value only at the 50%, while in the other two cases, the average values of *covrel* and OT wins are not significantly different. This confirms that there is not a clear trend of *covrel* improvement across checkpoints, and, since the final differences (i.e., at 100% of test cases) are much more pronounced in favor of *covrel*, it means that most of the work by *covrel* is done between 90% and 100%. This is likely due to a greater ability to uncover more subtle, rare, faults – a hypothesis also corroborated by observing that the average among the three means' differences "*covrel* - OT" in the configurations with "hard" faults is bigger than the same average with all the faults inside (23.07 *vs* 19.06).

C. How does covrel compare with traditional white-box coverage-based selection heuristics?

As we make use of coverage information to drive the selection phase of our approach, we performed further evaluations to investigate the effectiveness of *covrel* when compared with traditional white-box coverage-based selection. For doing so, we adopted the *greedy total* and *greedy additional* heuristics as they are often considered the baseline for coverage-based selection approaches due to their high cost-effectiveness ratio [36].

In making such a comparison, we need to consider though that test selection driven by reliability or by coverage set quite different targets. As said already, the former aims at selecting test cases so to reproduce usage in operation, whereas the latter aims at exploring exhaustively the program under test. Because of such different goals, the stopping criteria for reliability testing is achieving an acceptable estimation of reliability in operation, while the greedy coverage approaches, when applied for selection purposes, stop when the maximum

coverage that can be attainable by the set of tests available for selection is reached.

Therefore, the notion of operational profile and, consequently, the notion of fault importance, that we applied in the comparison between *covrel* and OT, is not applicable to traditional coverage-based selection heuristics. Thus it does not make sense to measure the delivered reliability at different points of the selection task. However, if the test suite derived by the selection approach is able to identify all the faults that could be possibly revealed, we can at least be sure that the maximum feasible reliability was achieved. Along these lines, to have comparable results, we performed the following steps:

1) We compute (from the 50 runs reported in Table II) the average number of test cases required by *covrel* to achieve the maximum attainable reliability;
2) For each combination of subject, version, fault-matrix and coverage criterion, we use the greedy total and greedy additional heuristics to derive 10 test suites targeting the maximum coverage that could be achieved by the subjects' test pool;
3) We stop the greedy selection when *(i)* all the faults are revealed or *(ii)* the maximum attained coverage is achieved, and we save the number of test cases selected.

In our experiments, in the vast majority of the observations ($\approx$ 97%) it was the case that either the selection heuristic finished the 10 iterations being able to achieve maximum reliability in all the cases, or it was never able to reveal all the faults before reaching the maximum achievable coverage. For the exceptional cases, if all the faults were revealed in at least 5 observations, we computed the average number of test cases required to achieve the maximum reliability; if not, we considered that the selection approach was not able to achieve maximum delivered reliability for that particular combination of subject, version, fault-matrix and coverage criterion.

TABLE V: Average number of test cases required by *covrel* and the greedy total heuristic to achieve the maximum attainable reliability

Subject	Function		Branch		Statement	
	covrel	*total*	*covrel*	*total*	*covrel*	*total*
Grep v1	**304**	547	**130**	-	**113**	574
Grep v2	**623**	-	**321**	531	**358**	501
Grep v3	**72**	-	**156**	487	**97**	477
Grep v4	**611**	728	**108**	724	**108**	728
Gzip v1	**178**	205	**118**	205	**124**	206
Gzip v2	26	3	28	1	29	3
Gzip v4	**25**	208	**26**	207	**25**	207
Gzip v5	**62**	204	**66**	207	**60**	206
Sed v2	**84**	-	114	92	**72**	88
Sed v3	**62**	-	**44**	352	**42**	352
Sed v4	**47**	-	**26**	123	**45**	131
Sed v5	**11**	-	**11**	362	**10**	362
Sed v6	**70**	-	**47**	104	**60**	89
Sed v7	**50**	-	**42**	-	**44**	-
Flex v1	**18**	411	**16**	379	**18**	382
Flex v2	**276**	669	**354**	669	**314**	664
Flex v3	**542**	-	**611**	614	622	618
Flex v4	96	4	228	4	257	4

Due to space limits, in Table V we report only the results of our evaluation when considering the *greedy total* approach and the default fault-matrix[9].

Precisely, Table V gives the average number of test cases required to achieve the maximum attainable reliability. For interpreting the results, the lower the number, the better. The cases in which the traditional coverage-based selection approach was not able to reveal all the faults is represented by a dash (–). This happened in 22% of the cases (12 out of 54) reported in Table V. The *covrel* approach performed better in 85% of the cases (46 out of 54). For ease of readability we highlight in bold the cases in which *covrel* performed better than the greedy total approach.

When considering the same setting, the greedy additional approach was not able to achieve the maximum attainable reliability in 41% of the cases (22 out of 54) and it was defeated by *covrel* in 50% of the cases.

D. Threats to validity

Beyond our best efforts in the accurate design and execution of experiments, our results might still suffer from validity threats.

As for **internal validity**, the selection of tests by *covrel* and operational profile-based technique includes in either case some random step, and thus without proper controlling the experiment settings, an observed difference in the outcomes might be due to random variability and not to actual differences in effectiveness. To prevent this, for each configuration we repeated the experiment 50 times, and in Sections V-A and V-B we reported the average outcomes over all repetitions.

A similar reasoning applies to our derivation of the operational profile partitions for each subject: these are used as proxies for true operational profiles, but if our approximation is not good, the observed results might not be related to reliability. Using averaged data over 50 executions corresponding to as many profiles helps to mitigate this risk, but further experiments with true profiles may be needed to fully neutralize it.

Another threat to internal validity might derive from the usage of the test suites available in the SIR repository [15], which: *i)* do not achieve full coverage, and *ii)* do not detect all faults in the repository. Therefore, we might have observed outcomes that are impacted by such characteristics of the test suites, rather than by the "treatment" under study. To mitigate this risk, we could have manipulated the test suite, e.g., by adding more test cases. However, the SIR repository data represent a "golden standard" for benchmarking purposes and are used in several similar studies. Thus, to make the study more objective and repeatable we preferred to undergo this potential threat and use each subject as it is provided with all of its artifacts.

Concerning **construct validity**, assessing which technique between *covrel* and OT reaches testing reliability faster (i.e.,

[9]Detailed results when considering the greedy additional approach and also the "hard" fault-matrix are available at: http://labsedc.isti.cnr.it/covrel2017.

finds all detectable failure regions) might not be a proper measure of effectiveness in reliability improvement. In real practice it would not be possible to conclude that reliability=1 has been reached because obviously we do not know a priori how many failure points exist. Thus other stopping rules to compare the "treatment" (*covrel*) against the baseline might produce different outcomes. To mitigate such threat, we fixed three checkpoints to complement the above result, and compared the techniques at such intermediate values. However, a true confirmation could only be achieved through reliability assessment either in production or under a real operational profile: in the context of the present study we could not perform either, but the experimentation of *covrel* within an industrial context is planned as future work.

Finally, concerning threats to **external validity**, the study results may suffer of representativeness of the used subjects and faults. With reference to subject representativeness, our study covered in total 18 variant versions of four C programs. Even though those 18 versions belong to four subjects only, it is well known that for those four subjects from SIR the differences between versions may be quite significant (and this is also visible from variations of results across versions in Figure 2). As for Sed, for instance, the development of v5 spans almost 5 years and the differences with respect to v4 are impressive — nearly every "major" function changed significantly. So, they could be considered as different programs. A similar case of fairly significant differences happened between versions v6 and v7 (as reported in the accompanying material from SIR). Nevertheless, before the results can be generalized, additional studies with a range of diversified subjects should be conducted.

With reference to faults representativeness, as said, in our study we considered seeded faults; subjects with real faults might yield different results. Control for this threat can be achieved only by conducting additional studies using subjects with real faults.

E. Verifiability

It is a goal of this study to support independent verification of the experiments, as well as repeatability with other subject programs, also in consideration of the discussed threats to external validity. To these aim, in addition to using subjects from the SIR repository we developed artifacts for automating the execution of the experiments, and we make them available to the scientific community[10].

The artifacts consist of:

- A prototypal *covrel* application with several modules written in the Java and Python programming languages; the prototype (delivered in a '.jar' file) takes as input the subject program and its version, and the desired number of repetitions (i.e., of profiles to generate);
- The detailed results of the experiments. These are packaged in a number of files in CSV format: each experiment includes the six configurations described in Section IV (three coverage criteria by two types of fault matrix), and produces five CSV files with many more details than those presented here, plus a textual file with the console output, which allows post-execution reconstruction of the progress of the experiment.

Operating instructions for running the prototype of *covrel* are provided in the README files accompanying it.

[10]The *covrel* artifacts and detailed results of this study are available at the URL: http://labsedc.isti.cnr.it/covrel2017.

VI. Conclusions

When product reliability is a major concern, software testing is often based on the operational profile, i.e., it exploits data on typical product usage so as to expose failures — and then remove the causing bugs — that will occur in operation with higher probability, thus contributing more to the delivered (un)reliability. One criticism to operational testing is that it pays too little attention to failures with low probability of occurrence, hence as testing proceeds reliability achieves some stable level that becomes difficult to improve further.

In this paper, we have introduced *covrel*, a reliability-driven adaptive testing technique that is hybrid, in that, first in its kind, it complements black-box operational profile information with white-box coverage measures based on count spectra. This way, test selection is dynamically adapted towards those program entities that have been less frequently covered, ultimately yielding improved reliability.

With regards to scalability of *covrel*, the cost of collecting coverage information is certainly a concern. The *covrel* technique is meant for practitioners who need to improve reliability beyond "saturation" of conventional operational testing: in such a context, the relevant cost is the high number of tests required to meet the reliability target, rather than the overhead of each test execution (namely, coverage costs). It is also worth noting that the use of count spectra (and associated cost) may compensate the availability of a not very detailed operational profile, which is a common situation and a known limitation of operational testing from the point of view of its practical industrial application. However, we cannot state that, in general, the benefits of *covrel* warrant coverage collection costs, as this will depend on trade-offs to be assessed on a case-by-case basis, and on how much the case under testing falls in the mentioned target context.

In the future, our aim is to adapt *covrel* for *test generation* (e.g. deriving the next test based on coverage spectra of tests executed up to a certain time) other than for *test selection* as we have done here. Moreover, further experiments with large-scale applications need to be conducted in order to fine-tune the adaptation criteria and of course to provide a more comprehensive assessment of the proposed technique.

Acknowledgment

This work has been partially supported by the GAUSS national research project, which has been funded by the MIUR under the PRIN 2015 program. Breno Miranda wishes to thank the Brazilian National Council for Scientific and Technological Development (CNPq) for providing his scholarship grant.

REFERENCES

[1] J. Musa, "Operational profiles in software-reliability engineering," *IEEE Software*, vol. 10, no. 2, pp. 14–32, 1993.

[2] M. Lyu, Ed., *Handbook of Software Reliability Engineering*. Hightstown, NJ, USA: McGraw-Hill, Inc., 1996.

[3] K.-Y. Cai, Y.-C. Li, and K. Liu, "Optimal and adaptive testing for software reliability assessment," *Information and Software Technology*, vol. 46, no. 15, pp. 989–1000, 2004.

[4] K.-Y. Cai, C.-H. Jiang, H. Hu, and C.-G. Bai, "An experimental study of adaptive testing for software reliability assessment," *Journal of Systems and Software*, vol. 81, no. 8, pp. 1406–1429, 2008.

[5] J. Musa, "Software reliability-engineered testing," *Computer*, vol. 29, no. 11, pp. 61–68, 1996.

[6] P. Frankl, R. Hamlet, B. Littlewood, and L. Strigini, "Evaluating testing methods by delivered reliability," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 586–601, 1998.

[7] L. Madani, C. Oriat, I. Parissis, J. Bouchet, and L. Nigay, "Synchronous testing of multimodal systems: an operational profile-based approach," in *Proc. 16th IEEE Int. Symposium on Software Reliability Engineering*, ser. ISSRE'05. IEEE, 2005, pp. 325–334.

[8] D. Cotroneo, R. Pietrantuono, and S. Russo, "RELAI testing: a technique to assess and improve software reliability," *IEEE Transactions on Software Engineering*, vol. 42, no. 5, pp. 452–475, 2016.

[9] B. Littlewood, P. T. Popov, L. Strigini, and N. Shryane, "Modeling the effects of combining diverse software fault detection techniques," *IEEE Transactions on Software Engineering*, vol. 26, no. 12, pp. 1157–1167, 2000.

[10] D. Cotroneo, R. Pietrantuono, and S. Russo, "Combining Operational and Debug Testing for Improving Reliability," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 408–423, 2013.

[11] B. Miranda and A. Bertolino, "Does code coverage provide a good stopping rule for operational profile based testing?" in *Proc. 11th Int. Workshop on Automation of Software Test*, ser. AST. ACM, 2016, pp. 22–28.

[12] M. Harrold, G. Rothermel, R. Wu, and L. Yi, "An empirical investigation of program spectra," *ACM SIGPLAN Notices*, vol. 33, no. 7, pp. 83–90, 1998.

[13] T. Ball, P. Mataga, and M. Sagiv, "Edge profiling versus path profiling: The showdown," in *Proc. 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL'98. ACM, 1998, pp. 134–148. [Online]. Available: http://doi.acm.org/10.1145/268946.268958

[14] G. Carrozza, R. Pietrantuono, and S. Russo, "Defect analysis in mission-critical software systems: a detailed investigation," *Journal of Software: Evolution and Process*, vol. 27, no. 1, pp. 22–49, 2015.

[15] "SIR: Software-artifact Infrastructure Repository." [Online]. Available: http://sir.unl.edu/portal/index.html

[16] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proc. 36th Int. Conference on Software Engineering*, ser. ICSE. ACM, 2014, pp. 435–445. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568271

[17] P. Kochhar, F. Thung, and D. Lo, "Code coverage and test suite effectiveness: Empirical study with real bugs in large systems," in *Proc. 22nd Int. Conference on Software Analysis, Evolution and Reengineering*, ser. SANER. IEEE, 2015, pp. 560–564.

[18] F. D. Frate, P. Garg, A. Mathur, and A. Pasquini, "On the correlation between code coverage and software reliability," in *Proc. 6th Int. Symposium on Software Reliability Engineering*, ser. ISSRE. IEEE, 1995, pp. 124–132.

[19] P. Frankl and Y. Deng, "Comparison of delivered reliability of branch, data flow and operational testing: A case study," *ACM SIGSOFT Software Engineering Notes*, vol. 25, no. 5, pp. 124–134, 2000. [Online]. Available: http://doi.acm.org/10.1145/347636.348926

[20] D. Alrmuny, "A Comparative Study of Test Coverage-Based Software Reliability Growth Models," in *Proc. 11th Int. Conference on Information Technology: New Generations*, ser. ITNG. IEEE, 2014, pp. 255–259.

[21] M.-H. Chen, M. Lyu, and W. Wong, "Effect of Code Coverage on Software Reliability Measurement," *IEEE Transactions on Reliability*, vol. 50, no. 2, pp. 165–170, 2001.

[22] D. Leon and A. Podgurski, "A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases," in *Proc. 14th Int. Symposium on Software Reliability Engineering*, ser. ISSRE. IEEE, 2003, pp. 442–453. [Online]. Available: http://dl.acm.org/citation.cfm?id=951952.952367

[23] B. Miranda and A. Bertolino, "Scope-aided Test Prioritization, Selection and Minimization for Software Reuse," *Journal of Systems and Software*, vol. doi:10.1016/j.jss.2016.06.058, 2016.

[24] T. Ball and J. R. Larus, "Using paths to measure, explain, and enhance program behavior," *Computer*, vol. 33, no. 7, pp. 57–65, 2000.

[25] T. Reps, T. Ball, M. Das, and J. Larus, "The use of program profiling for software maintenance with applications to the year 2000 problem," *ACM SIGSOFT Software Engineering Notes*, vol. 22, no. 6, pp. 432–449, 1997. [Online]. Available: http://dx.doi.org/10.1007/3-540-63531-9_29

[26] T. Xie and D. Notkin, "Checking inside the black box: regression testing by comparing value spectra," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 869–883, 2005.

[27] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. Van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780–1792, 2009.

[28] W. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[29] J. Lv, B.-B. Yin, and K.-Y. Cai, "On the asymptotic behavior of adaptive testing strategy for software reliability assessment," *IEEE Transactions on Software Engineering*, vol. 40, no. 4, pp. 396–412, 2014.

[30] P. Frankl, R. Hamlet, B. Littlewood, and L. Strigini, "Evaluating testing methods by delivered reliability," *IEEE Transactions on Software Engineering*, vol. 24, no. 8, pp. 586–601, 1998.

[31] C. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. New York, NY, USA: Springer-Verlag, 2006.

[32] D. Fox, "Adapting the Sample Size in Particle Filters Through KLD-Sampling," *Int. Journal of Robotics Research*, vol. 22, no. 12, pp. 985–1003, 2003.

[33] M. Sridharan and A. Namin, "Prioritizing Mutation Operators Based on Importance Sampling," in *Proc. 21st Int. Symposium on Software Reliability Engineering*, ser. ISSRE, 2010, pp. 378–387.

[34] B. Yang and M. Xie, "A study of operational and testing reliability in software reliability analysis," *Reliability Engineering & System Safety*, vol. 70, no. 3, pp. 323 – 329, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832000000697

[35] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[36] D. Hao, L. Zhang, L. Zhang, G. Rothermel, and H. Mei, "A Unified Test Case Prioritization Approach," *ACM Transactions on Software Engineering and Methodology*, vol. 24, no. 2, pp. 10:1–10:31, 2014. [Online]. Available: http://doi.acm.org/10.1145/2685614