On the Fault-Detection Capabilities of Adaptive Random Test Case Prioritization: Case Studies with Large Test Suites (PDF)

Article · January 2012

DOI: 10.1109/HICSS.2012.454

CITATIONS
17

READS
146

3 authors:

Zhi Quan Zhou
University of Wollongong
69 PUBLICATIONS **1,287** CITATIONS

SEE PROFILE

Arnaldo Sinaga
Institut Teknologi Del, Indonesia
11 PUBLICATIONS **36** CITATIONS

SEE PROFILE

Willy Susilo
University of Wollongong
677 PUBLICATIONS **11,421** CITATIONS

SEE PROFILE

On the Fault-Detection Capabilities of Adaptive Random Test Case Prioritization: Case Studies with Large Test Suites *

Zhi Quan Zhou†, Arnaldo Sinaga‡ and Willy Susilo
School of Computer Science and Software Engineering
University of Wollongong
Wollongong, NSW 2522, Australia

***Abstract*—An adaptive random (AR) testing strategy has recently been developed and examined by a growing body of research. More recently, this strategy has been applied to prioritizing regression test cases based on code coverage using the concepts of Jaccard Distance (JD) and Coverage Manhattan Distance (CMD). Code coverage, however, does not consider frequency; furthermore, comparison between JD and CMD has not yet been made. This research fills the gap by first investigating the fault-detection capabilities of using frequency information for AR test case prioritization, and then comparing JD and CMD. Experimental results show that "coverage" was more useful than "frequency" although the latter can sometimes complement the former, and that CMD was superior to JD. It is also found that, for certain faults, the conventional "additional" algorithm (widely accepted as one of the best algorithms for test case prioritization) could perform much worse than random testing on large test suites.**

I. Introduction

Software testing is an involved task and normally accounts for over half of the total cost in typical software development projects [1], [2], [3]. Any improvement in the cost-effectiveness of testing, therefore, will directly improve the overall cost-effectiveness of software processes.

Among the various software testing techniques, *Random Testing* (RT) is the most fundamental strategy [4]. RT is simple in concept. It is often easy to apply, can often exercise the software under test in unexpected ways, and has demonstrated effectiveness in detecting failures [5]. Furthermore, in situations where source code and specifications of the software under test are unavailable or incomplete, RT may be the only practical approach to testing the software. RT has been widely applied for decades and forms the basis of many other testing techniques [3], [6], [7], [8], [9], [10], [11], [12]. For example, in partition testing, an equivalence class normally contains a large number of elements. It is often through random selection that one of the elements is chosen as a test case to represent the entire equivalence class.

* This project was supported in part by a linkage grant of the Australian Research Council (Project ID: LP100200208) and a UIC International Links Grant of the University of Wollongong.

† All correspondence should be addressed to Dr. Zhi Quan Zhou, School of Computer Science and Software Engineering, University of Wollongong, Wollongong, NSW 2522, Australia. Email: zhiquan@uow.edu.au. Telephone: (61-2) 4221 5399.

‡ Arnaldo Sinaga is also with Del Polytechnic of Informatics, North Sumatra, Indonesia.

In software testing, if an executed test case does not reveal a failure, it is called a *successful test case*. Conventionally, successful test cases are considered useless since they did not reveal any failure [13], and they are retained mainly for regression testing. It is noted, however, that successful test cases do carry useful information [14], [15], [16], such as clues about the location of failure regions. For example, in situations where failure-causing inputs are clustered in one or a few regions of the input domain, selecting a test case that is very close to a successful test case is intuitively not wise. This kind of location information of successful test cases has been employed to develop a testing strategy known as *Adaptive Random Testing* (ART) [5], [14], [17], [18], [19]. Compared with RT, test cases generated by ART are farther apart from each other and, hence, they are more evenly spread over the input domain. ART has demonstrated fault-detection capability in terms of using fewer test cases to detect the first failure than RT. Jaygarl et al. [20] noted that ART "became one of the most effective approaches in the automatic test generation area."

ART was initially applied to programs with numerical input types. This is because the distance between numerical inputs can be directly calculated using the Euclidean measure. Ciupa et al. [19] applied ART to object-oriented software by developing a measure for computing the distance between objects. Jaygarl et al. followed up on this work for object-oriented software testing [20].

More recently, Jiang et al. [21] and Zhou [22] proposed the Jaccard distance and Manhattan distance, respectively, to enable ART to be applied to any arbitrary program without any restriction on input types, provided that certain coverage information is available before testing, such as previous statement or branch coverage information of test cases in the context of regression test case prioritization. It should be noted, however, that coverage does not take account of frequency. In other words, frequencies 1 and 100 are considered identical in Jiang et al.'s and Zhou's coverage-based distance calculations [21], [22]. Our first research question or objective is, therefore, to propose frequency-based distance measures and compare their fault-detection capabilities against those of coverage-based measures. Our second research question is to compare the fault-detection capabilities of the Jaccard-distance-based ART and Manhattan-distance-

based ART. Answers to these two research questions will provide important guidance to software testing practitioners.

The rest of this paper is organized as follows: Section II provides background information on ART with a focus on the FSCS-ART algorithm, Jiang et al.'s work [21] and Zhou's work [22]. Section III describes the design of experiments and analyzes the experimental results to address the first research question, and Section IV addresses the second research question. Section V concludes the paper.

II. Background

The *adaptive random testing* (ART) strategy was proposed as an improvement of random testing (RT) with an objective of detecting failures early [14]. The basic idea is to more evenly spread test cases over the input domain. Different ART algorithms have been proposed to implement the concept of "even spread." The first ART algorithm is known as the Fixed Size Candidate Set (FSCS-ART) algorithm [18], [23]. Because of its simplicity, this algorithm has been adopted in many ART-related studies. This algorithm maintains a set E that stores all test cases that have already been executed, and a set C that stores test case candidates. Initially, a test case is randomly selected from the input domain and run. If no failure is detected, it will be added to E. Then a fixed number (normally 10) of test case candidates are randomly sampled from the input domain and put to C. For each candidate c_i in C, its distance to E is measured. To measure the distance from c_i to E, the distance between c_i and each element in E is calculated, and the minimum one among all these distances is taken as the distance from c_i to E. The candidate c_j whose distance to E is the maximum is chosen to be the next test case, and all the other candidates are discarded. Intuitively, test cases thus selected are far apart from each other. This process is repeated until the testing stopping criterion is met. The time complexity of this algorithm is in $O(n^2)$ where n is the number of test cases executed. It has been demonstrated that FSCS-ART can achieve a much lower *F-measure* than RT, where F-measure is the number of test cases executed to detect the first failure [14].

It should be noted that an ART sequence of test cases can be generated off-line prior to test case execution. In other words, test case generation of ART (in $O(n^2)$) can be conducted in parallel with other software engineering activities even before the current version of the software under test has been developed. On the other hand, test case execution and result verification must be conducted after the software has been developed. In other words, compared with test case generation, the activities of test case execution and result verification are more critical in the entire software development life cycle because any delay of them will probably result in a delay of the entire project. ART effectively reduces the time required for these two critical activities.

To enable ART to be applied to all kinds of programs regardless of their input types, Jiang et al. [21] proposed to use the Jaccard Distance. The JD between two test cases a and b is given by $JD(a,\ b) = 1 - |A \cap B|/|A \cup B|$, where A and B are the sets of statements (or branches, functions, etc) covered by a and b, respectively. Furthermore, instead of using a predetermined number of candidates as in the FSCS-ART algorithm, Jiang et al.'s algorithm builds the candidate set using a flexible number of elements: the set is built by iteratively adding a randomly sampled candidate until the new candidate cannot increase program coverage or the candidate set is full. Therefore, the numbers of candidates can vary in each round of test case generation. In this paper we will refer to this algorithm as the Flex-ART Algorithm, as opposed to the FSCS-ART algorithm.

Using the above methods and by employing different types of coverage information (such as statement, branch and function coverages), Jiang et al. developed a family of ART algorithms for regression test case prioritization and conducted empirical evaluations on their fault-detection capabilities against the Average Percentage Faults Detected (APFD) measure [24]. They found that "ART-br-maxmin is the most effective technique in the proposed family of ART techniques" where "ART-br-maxmin" is the ART algorithm employing branch coverage information, and "maxmin" means to maximize the minimum distance between a candidate and all the executed test cases as explained previously. Jiang et al. also reported that the APFD of ART-br-maxmin was not only statistically superior to RT but also "consistently comparable to some of the best coverage-based prioritization techniques (namely, the 'additional' techniques) and yet involves much less time cost."

Independent of Jiang et al.'s work [21], Zhou proposed a Coverage Manhattan Distance (CMD) for ART [22]: let a be a test case, and $v_a = (a_1, a_2, \ldots, a_n)$ be a vector that records certain type of coverage information (for example, statement coverage) of a, where $a_i \in \{0,\ 1\}$ for $i = 1, 2, \ldots, n$, and n is the total number of elements under consideration. In the case of statement coverage, for example, n is the total number of statements in the program, and $a_i = 1$ if and only if the ith statement of the program has been touched at least once during the execution of a; otherwise $a_i = 0$. Let b be another test case and let $v_b = (b_1, b_2, \ldots, b_n)$ store the coverage information of b. The CMD between a and b is defined as follows:

$$CMD(a,\ b) = \sum_{i=1}^{n} |a_i - b_i|. \tag{1}$$

In the context of regression testing, v_a and v_b can be taken from previous test results, such as statement/branch/function coverage data collected when the test cases were run on a previous version of the program under test [24]. Also note

that calculating JD and calculating CMD have the same time complexity. It was found that the F-measure of CMD-based ART is much smaller than that of RT [22].

To the best of the authors' knowledge, no empirical studies have been reported to compare the effectiveness of JD- and CMD-based ART methods. Neither is there any study on the use of frequency (rather than coverage) information for ART. These studies will be reported in this paper.

III. Comparing Coverage- and Frequency-Based Distance Measures

A. Frequency-Based Distance Measures

Let $E_u = (u_1, u_2, \ldots, u_n)$ be an execution profile of test case u, where u_i ($i = 1, 2, \ldots, n$) is the number of times (that is, frequency) that the ith construct of the program is exercised by u. Similarly, let $E_v = (v_1, v_2, \ldots, v_n)$ be an execution profile that records the frequency information of test case v. We define the *frequency Manhattan distance* (FMD) between u and v as

$$FMD(u, v) = \sum_{i=1}^{n} |u_i - v_i| \tag{2}$$

and the *frequency Hamming distance* (FHD) between u and v as

$$FHD(u, v) = \sum_{i=1}^{n} k_i \tag{3}$$

where $k_i = 0$ if $u_i = v_i$; otherwise $k_i = 1$, $i = 1, 2, \ldots, n$. Obviously, FHD concerns how many (u_i, v_i) pairs are not identical regardless of how large the difference is; whereas FMD sums up the differences between each (u_i, v_i) pair.

B. Design of the Experiments

To evaluate and compare the fault-detection capabilities of the proposed distance measures, empirical studies have been conducted. Two subject programs, which are widely used in the literature of software engineering research, have been adopted to conduct the experiments. The first subject program is *replace*, which is the largest and most complex program in the *Siemens suite of programs* [25], downloaded from http://pleuma.cc.gatech.edu/aristotle/Tools/subjects/. The *replace* program performs regular expression matching and substitutions. It has 512 lines of C code (excluding blanks and comments) and 20 functions. The *replace* package also includes 32 faulty versions that cover the most varieties of logic errors among the Siemens suite of programs [26]. A total of 5,542 test cases are also included in the *replace* package. We excluded faulty versions 13, 23, 26 and 32 because they either generated identical outputs as the base version or are not stable in that they generated different outputs when run at different times or under different environments.

The second subject program is *space*, downloaded from the *Software-artifact Infrastructure Repository* (http://sir.unl.edu) [27]. The *space* program is an interpreter for an array definition language. It consists of 6,199 lines of C code (excluding blanks and comments) and 136 functions. The *space* package includes 38 faulty versions. According to the Software-artifact Infrastructure Repository, the faults in these faulty versions are real faults discovered during the development of the program. Faulty versions 1, 2, 32 and 34 are excluded from our experiments as they generated identical outputs as the base version. The *space* package includes 13,551 test cases.

For each subject program, the experiment was conducted by first running all the test cases on the base version, and the Unix utility *gcov*, which is a standard test coverage tool in concert with *gcc*, was used to record statement and branch coverage and frequency data of every test case. Outputs of the base version were recorded to serve as the test oracle.

Then, for each faulty version, 7 types of test case sequences (permutations of the entire test pool) were generated. These 7 types of sequences are: pure random sequence, adaptive random (AR) sequence using the branch CMD measure, AR sequence using the branch FMD measure, AR sequence using the branch FHD measure, AR sequence using the statement CMD measure, AR sequence using the statement FMD measure, and AR sequence using the statement FHD measure. To obtain statistically meaningful results, we generated 1,000 different instances for each of these 7 sequence types. Therefore, for each faulty version, a total of $7 \times 1,000 = 7,000$ test case sequences were generated. Following the common practice in regression testing [24], the coverage and frequency data were collected from test case executions on the base version rather than the faulty versions. A failure is revealed when output of the faulty version differs from that of the base version. For each faulty version and each of the 7 sequence types, the mean F-measure of the 1,000 trials (that is, the 1,000 instances of the sequence type) was calculated.

C. Validity of Using Large Test Pools in Experiments

It should be noted that, different from previous work (e.g. [21], [24]) where test case prioritizations were conducted only on subsets of the given test pool, this research conducts test case prioritizations on the *entire* test pool. We believe that an empirical study using the entire test pool (of a large size) is more meaningful than a study using only subsets (with a much smaller size) of the pool. After all, if the size of the test suite is small, there would be little need for test case prioritization techniques. In real life, the size (number of test cases) of a regression test suite can often become much larger than the size (number of statements) of the program under test. Rothermel et al., for example, reported that "Running all of the test cases in a test suite, however, can require a large amount of effort. For example, one of our

industrial collaborators reports that for one of its products of about 20,000 lines of code, the entire test suite requires seven weeks to run" [24]. At Microsoft, a test set includes "millions of test cases" during a re-test [28]. The popular SQL database engine SQLite (as of version 3.7.5) "consists of approximately 73.0 KSLOC of C code" but "has 1251 times as much test code and test scripts – 91378.6 KSLOC" together with "Millions and millions of test cases" [29].

D. Results of Experiments

1) Results of Experiments with the Replace Program: Table I shows the experimental results with the *replace* and *space* programs. Column #2 shows statistics of the observed mean F-measure of pure random testing (OFRT for short). For each faulty version of the program under test, 1,000 random sequences were generated and, hence, 1,000 F-measures were observed and their mean was calculated to give the OFRT value of that faulty version. For succinctness of presentation, raw results of each individual faulty version are not shown in Table I ; instead, statistics over all the faulty versions are given in the table, namely average (Ravg and Savg for *replace* and *space* programs, respectively), total (Rttl and Sttl for *replace* and *space*, respectively), maximum (Rmax and Smax for *replace* and *space*, respectively), and minimum (Rmin and Smin for *replace* and *space*, respectively). Column #3 corresponds to the theoretical mean F-measure of pure random testing (TFRT for short) calculated using the formula given in Zhou [22]. Note that the pure random testing process is through sampling *without* replacement.

For the 28 faulty versions of the *replace* program, the relative errors between OFRT and TFRT scores (given by $\frac{|\text{TFRT-OFRT}|}{\text{TFRT}}$) ranged from 0.01% (for version 5) to 5.17% (for version 11), with mean 2.04%. The difference between the two sample means is 2.99%, with a relative error of 1.29%, which is quite small. A paired-samples t test returns a p-value of .08 (two-tailed), greater than the .05 significance level. In summary, the difference between the two sample means of OFRT and TFRT is neither large nor statistically significant.

Columns #4, #5 and #6 of Table I show statistics of the mean F-measures of ART based on CMD, FMD and FHD, respectively, that employs branch coverage information. Furthermore, for each faulty version, we calculate the following ratios: R_CMD (=CMD/TFRT), R_FMD (=FMD/TFRT), and R_FHD (=FHD/TFRT). Obviously, the smaller the ratio, the better the fault-detection capability. Columns #7, #8 and #9 list statistics of these ratios, and those greater than 1 are highlighted in the table, indicating situations where pure random testing outperformed ART. Note the difference between "avg" and "ttl": for example, the cell at row "Ravg" and column #7 is 61.04%; whereas that at row "Rttl" in the same column is 59.06%. This is because the former gives the *average* ratio, that is the *mean* of all the 28 R_CMD values (from all of the 28 faulty versions of *replace*); whereas the latter gives the total or *overall* ratio, calculated as Rttl of CMD (3839.97, see column #4) divided by Rttl of TFRT (6501.33, see column #3).

Consider the R_CMD scores, whose statistics are shown in column #7 of Table I . For the *replace* program, ART was superior to RT in all of the faulty versions except versions 6, 17 and 20 – these 3 versions have an R_CMD score slightly higher than 100%. For all of the 28 faulty versions, the R_CMD scores ranged between a minimum of 28.05% and a maximum of 104.73%, with an average of 61.04% (and an overall of 59.06%). This means that ART using branch CMD required in average 38.96% (or in total 40.94%) fewer test cases than RT to detect the first failure, which is a considerable saving.

A paired-samples t test is further conducted to compare the sample means of the F-measures (for TFRT and branch CMD) over the 28 faulty versions of *replace*. A p-value of .026 (two-tailed) is returned, which is below the .05 significance level. In conclusion, compared with RT, branch CMD achieved both large (40.94%) and statistically significant savings; branch CMD also had the smallest standard deviation in its F-measures among all the algorithms listed in Table I and, hence, it was the most stable algorithm.

Next, consider the R_FMD scores for the *replace* program, whose statistics are shown in column #8 of Table I . While FMD outperformed RT in most of the faulty versions, the performance of FMD was surprisingly poor for version 8 (R_FMD=852.60%) and version 15 (R_FMD=3046.75%). These two extremely large values resulted in a large average score of 180.99%. However, the Rttl score is not too bad, which is 99.54%. This means that ART based on branch FMD used a similar total number of test cases as RT. We also noticed that the standard deviation of R_FMD over all the 28 faulty versions is as large as 582.30%. In comparison, that of R_CMD is only 25.44%. This indicates that branch CMD was much more stable.

It is further observed that branch FMD can be complementary to branch CMD: for certain faults, when the latter is large, the former is quite small. For example, for version 22 of *replace*, the mean CMD F-measure is 256.12 whereas that for FMD is only 56.66; for version 19, the mean CMD F-measure is 557.81 whereas that for FMD is only 187.80. Similar observations are also made to several other versions of *replace*. Further studies on conditions under which FMD is (in)effective and how frequency information can be applied to complement CMD are beyond the scope of the present paper. These are interesting future research topics.

Analysis of the branch FHD results is similar to that of FMD, except that FHD performed slightly better than FMD. There are still two very large R_FHD scores, namely 402.58% for version 8 and 2499.66% for version 15, which resulted in a large average score of 144.62%. The Rttl score

Table I
RESULTS OF EXPERIMENTS (1,000 TRIALS WERE CONDUCTED TO EACH AND EVERY FAULTY VERSION.)
OFRT: OBSERVED MEAN F-MEASURE OF RT; TFRT: THEORETICAL MEAN F-MEASURE OF RT.

	F-measure of RT		ART (branch)						ART (statement)					
	OFRT	**TFRT**	**CMD**	**FMD**	**FHD**	**R_CMD**	**R_FMD**	**R_FHD**	**CMD**	**FMD**	**FHD**	**R_CMD**	**R_FMD**	**R_FHD**
Ravg	229.20	232.19	137.14	231.11	192.10	61.04%	180.99%	144.62%	158.48	219.35	196.34	70.53%	165.78%	157.17%
Rttl	6417.67	6501.33	3839.97	6471.21	5378.93	59.06%	99.54%	82.74%	4437.36	6141.81	5497.55	68.25%	94.47%	84.56%
Rmax	1381.73	1385.75	777.45	2768.55	2271.41	104.73%	3046.75%	2499.66%	1157.40	2585.50	2407.51	122.65%	2845.31%	2649.43%
Rmin	18.22	17.88	6.72	5.96	5.03	28.05%	13.55%	19.34%	8.98	6.60	5.61	26.90%	13.97%	16.76%
Savg	93.99	93.49	29.06	97.43	34.36	54.78%	111.74%	58.08%	35.82	89.33	42.51	60.14%	103.96%	62.72%
Sttl	3195.66	3178.53	988.01	3312.53	1168.17	31.08%	104.22%	36.75%	1217.98	3037.33	1445.27	38.32%	95.56%	45.47%
Smax	406.11	410.67	129.14	664.41	164.37	110.69%	369.40%	125.54%	213.05	609.07	258.59	143.06%	313.56%	173.64%
Smin	1.06	1.07	1.05	1.05	1.06	16.68%	24.31%	14.93%	1.05	1.05	1.05	18.59%	26.56%	20.24%
#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15

is 82.74%, indicating that ART required a smaller total number of test cases than RT. The standard deviation of R_FHD over all of the 28 faulty versions is 466.76%, which is very high.

It is interesting to observe that the mean F-measures of FHD actually outperformed those of CMD in all of the faulty versions of *replace* except versions 8, 15 and 24. This observation indicates that frequency information is indeed useful and, if employed properly, can complement coverage-based measures.

The last 6 columns (#10 to #15) show statistics of experimental results of ART using statement coverage/frequency information. It is found that there is a strong and significant positive correlation between branch-based and statement-based ART scores: based on the results of the 28 faulty versions, the correlation coefficients between the pairs (R_CMD (branch), R_CMD (statement)), (R_FMD (branch), R_FMD (statement)), and (R_FHD (branch), R_FHD (statement)) are .821, .996 and .998, respectively ($p < .001$). Furthermore, branch CMD (column #7) had a better average performance than statement CMD (column #13), and the difference is statistically significant (a paired-samples t test returns a p-value of .006, two-tailed, below the .05 significance level). The paired-samples t test, however, shows that there is no statistically significant difference between the sample means of R_FMD (branch) and R_FMD (statement) ($p = .25$), or between the sample means of R_FHD (branch) and R_FHD (statement) ($p = .14$).

In summary, experimental results with the *replace* program show that the coverage-based measure (namely, CMD) was better than the frequency-based measures (namely, FMD and FHD) as well as RT, and branch CMD was better than statement CMD. It is also revealed that frequency information is useful and can complement the coverage-based measure.

2) Results of Experiments with the Space Program: A total of 34 faulty versions of the *space* program were used in the experiments. The relative errors between OFRT and TFRT scores ranged from 0.04% (for version 35) to 5.64% (for version 9), with a mean of 1.87%. The difference between the Savg of OFRT and the Savg of TFRT is only 0.50 ($= 93.99 - 93.49$, see Table I). A paired-samples t test for OFRT and TFRT over the 34 faulty versions returns a p-value of .48 (two-tailed). In conclusion, the difference between the sample means of OFRT and TFRT is very small and not statistically significant.

Consider the branch R_CMD scores of the *space* program, whose statistics are shown in the last 4 rows of column #7 of Table I . Out of the 34 faulty versions, branch CMD outperformed RT in 33 versions; only for version 3, branch CMD spent more (10.69%) test cases than RT. The best performance of branch CMD was achieved for version 22 (R_CMD = 16.68%). The average R_CMD score for all versions is 54.78%, and the overall score is 31.08%, hence giving an overall saving of 68.92%. A paired-samples t test for TFRT and branch CMD F-measures over the 34 faulty versions was conducted, which returns a p-value of .001 (two-tailed). In summary, compared with RT, ART based on branch CMD achieved both large and statistically significant savings.

Consider the branch R_FMD scores whose statistics are shown in the last 4 rows of column #8 of Table I . It is observed that 14 (out of 34) faulty versions of *space* had an R_FMD score higher than 100%. The average R_FMD score is 111.74% and the overall score is 104.22%, both of which indicate that branch FMD did not gain savings. To compare the mean F-measures of FMD and TFRT, a paired-samples t test over the 34 faulty versions was conducted and a p-value of .843 (two-tailed) was returned, which indicates that the difference between the mean F-measures of branch FMD and TFRT is not statistically significant. It is also observed that, in 5 faulty versions of *space*, branch FMD outperformed branch CMD, which confirms that frequency information can complement the coverage-based measures for certain kinds of faults.

The last 4 rows of column #9 of Table I show some

statistics of the branch R_FHD scores, which are much better than the branch R_FMD scores. It is observed that only 2 (out of 34) faulty versions of *space* had a branch R_FHD score higher than 100%, namely version 3 (125.54%) and version 36 (110.37%). The sample means of R_FHD and R_CMD over the 34 faulty versions of *space* are compared using a paired-samples t test, which returns a p-value of .062 (two-tailed), greater than the .05 significance level – this indicates that the difference of sample means between R_FHD and R_CMD is not statistically significant (or only of marginal significance). In comparison, the sample means of R_FMD and R_CMD differ significantly (p = .001, two-tailed). The FHD F-measures are also significantly better than the TFRT scores (p = .002, two-tailed).

The last 6 columns (#10 to #15) show the results of ART based on statement coverage/frequency information. For the 34 faulty versions of *space*, it is again observed that there is a strong and significant positive correlation between branch-based and statement-based ART scores. The correlation coefficients between the pairs (R_CMD (branch), R_CMD (statement)), (R_FMD (branch), R_FMD (statement)), and (R_FHD (branch), R_FHD (statement)) are .945, .968 and .926, respectively ($p < .001$). Furthermore, branch-based CMD and FHD (columns #7 and #9) had a better average performance than their statement-based counterparts (columns #13 and #15), and the difference is statistically significant (34 faulty versions, $p = .008$ for CMD and $p = .033$ for FHD, two-tailed). Statistically significant difference between branch R_FMD and statement R_FMD, however, is not observed ($p = .061 > .05$, two-tailed).

In summary, experimental results with the *space* program confirm that ART based on CMD was the best algorithm, and branch CMD was better than statement CMD. It is also confirmed that frequency information can complement the coverage-based measure in certain situations. These findings are consistent with those obtained from the *replace* program experiments.

IV. Comparing JD and CMD

Section III addressed the first research question raised in Section I and found that CMD outperformed the frequency-based distance measures, namely, FMD and FHD. This section will compare CMD and JD (Jaccard Distance) to answer the second research question, and FMD and FHD will not be included in the study as they have been shown to be less effective than CMD. Furthermore, this section will focus on ART based on *branch* coverage, which has been shown to be more effective than that based on statement coverage.

A. Design of the Experiments

As introduced in Section II , to generate the candidate set of test cases, there are two algorithms, namely, the conventional FSCS-ART algorithm and Jiang et al.'s Flex-ART algorithm. Therefore, there are four combinations of algorithms under investigation, written as cmd-10, jd-10, cmd-flex, and jd-flex, representing CMD-based FSCS-ART, JD-based FSCS-ART, CMD-based Flex-ART, and JD-based Flex-ART algorithms, respectively, where cmd-10 was proposed in Zhou [22] (and "10" refers to the size of the candidate set) and jd-flex was proposed in Jiang et al. [21]. In addition, we also include pure random testing (RT) and the Additional algorithm (*add* for short) in our comparison, where RT serves as a benchmark, and *add* is one of the best known coverage-based test case prioritization algorithms [21], [24]. Therefore, there is a total of 6 algorithms (or prioritization methods) under investigation.

The *replace* and *space* programs were used for empirical evaluations. The experiments were conducted in a similar way as described in Section III . For each of these two subject programs and for each of the 6 prioritization methods, 1,000 permutations of the entire test pool were generated. For RT, the 1,000 permutations are random sequences without referring to coverage data. For *add*, it is a deterministic algorithm and, theoretically, it should generate identical sequences for the same test suite. However, because of the randomness in tie breaking, different sequences may also be generated. Each sequence (permutation) of test cases were run on each faulty version, with its F-measure and APFD value recorded. The mean F-measure and mean APFD value of the 1,000 trials were calculated for each faulty version and each prioritization method.

Experimental results on F-measures and APFD values are summarized in Table II , where a smaller F-measure or a larger APFD value indicates a higher fault-detection capability. For succinctness of presentation, raw results of each individual faulty version are not shown in the table; nevertheless, the analysis of these results will be presented. RT in Table II corresponds to OFRT in Table I , and cmd-10 in Table II corresponds to branch-CMD in Table I .[1]

It is observed that the differences between the APFD values of different techniques are quite small. We found that this is because the sizes of the test suites are large compared with the sizes (numbers of statements) of the programs under test. Nevertheless, when analyzed against the mean F-measures, different techniques show large differences.

B. Results of Experiments with the Replace Program

1) Comparing with RT in terms of F-measure: First, compare the pair (RT, *add*). Out of the 28 faulty versions of *replace*, *add* outperformed RT on 22 versions. As shown in Table II , the mean of mean F-measures of *add* is 186.01, which is 21.02% lower than that of RT (235.52).

[1]Note that their respective F-measure results as shown in Tables I and II are not identical because of the randomness of the algorithms, for the experiments of Section III and those of Section IV were conducted independent of each other.

Table II
RESULTS OF EXPERIMENTS (1,000 TRIALS WERE CONDUCTED TO EACH AND EVERY FAULTY VERSION BASED ON BRANCH COVERAGE.)

	RT	add	cmd-10	jd-10	cmd-flex	jd-flex
Mean of the 28 mean F-measures for *replace*	235.52	186.01	132.07	159.91	149.32	168.32
Mean APFD for *replace*	0.9600	0.9665	0.9771	0.9718	0.9743	0.9707
Mean of the 34 mean F-measures for *space*	94.04	62.29	29.68	37.41	34.83	35.14
Mean APFD for *space*	0.9931	0.9974	0.9979	0.9973	0.9975	0.9975

These observations suggest that *add* outperformed RT. A paired-samples t test over the 28 mean F-measures (of the 28 faulty versions), however, returned a p-value of .672 (two-tailed), which indicates that there is no statistically significant difference between the sample means of RT and *add*. Furthermore, it is found that, to detect the first failure for version 15 of the *replace* program, *add* used 2,628.99 test cases in average, whereas RT used only 93.89 test cases in average (and the theoretical mean F-measure of RT for this faulty version is 90.87 [22]). This observation indicates that while in most cases *add* can use fewer test cases to detect the first failure than RT, the former can sometimes become very unreliable as 2,628.99 is 29 times as large as 90.87. To the best of the authors' knowledge, this problem of the Additional algorithm has not been previously reported in the literature.

The CMD-based ART methods, namely, cmd-10 and cmd-flex, outperformed RT on 26 and 27 versions, respectively, out of the 28 faulty versions of *replace*, with respective mean values of 132.07 (43.92% lower than that of RT) and 149.32 (36.60% lower than that of RT). A paired-samples t test for RT and cmd-10 over the 28 faulty versions returned a p-value of .028$<$.05, and that for RT and cmd-flex returned a p-value of .022$<$.05, indicating that both cmd-10 and cmd-flex outperformed RT significantly.

The two JD-based ART methods also performed better than RT, but not as good as the CMD-based methods: out of the 28 faulty versions of *replace*, jd-10 and jd-flex outperformed RT on 17 and 18 versions, respectively, with respective mean values of 159.91 (32.10% lower than that of RT) and 168.32 (28.53% lower than that of RT). Paired-samples t tests for (RT, jd-10) and for (RT, jd-flex) returned p-values of .069$>$.05 and .048$<$.05, respectively.

In summary, out of all the investigated methods, the two CMD-based ART methods (cmd-10 and cmd-flex) outperformed RT most significantly, and cmd-10 achieved the highest saving in terms of mean F-measure.

2) Comparing with the Additional algorithm in terms of F-measure: Out of the 28 faulty versions of *replace*, the ART methods, namely, cmd-10, jd-10, cmd-flex and jd-flex, outperformed *add* for 12, 9, 11 and 8 versions, respectively. In this sense, *add* has shown a better performance than ART. It is found, however, that all these 4 ART methods outperformed *add* in terms of mean of mean F-measures (that is, average mean F-measures) and mean APFD. Therefore, ART can be considered to have outperformed *add*. Paired-samples t tests for each of the 4 pairs of (*add*, ART method) over the 28 faulty versions returned a p-value greater than or equal to .583, indicating that the differences between the sample means of *add* and the ART methods are not statistically significant.

Among all of the 6 methods, cmd-10 had the best (lowest) mean of mean F-measures and the best (highest) mean APFD, as shown in Table II .

3) Comparing CMD and JD in terms of F-measure: A main objective of this research is to compare the fault-detection capabilities of CMD and JD in the context of ART. Out of the 28 faulty versions of *replace*, cmd-10 outperformed jd-10 on 24 versions, with an average mean value 17.41% ($= 1 - 132.07/159.91$) lower than that of jd-10; cmd-flex outperformed jd-flex also on 24 versions, with an average mean value 11.29% lower than that of jd-flex. A paired-samples t test for cmd-10 and jd-10 over the 28 faulty versions returned a p-value of .005, and that for cmd-flex and jd-flex returned a p-value of .011, indicating that the two CMD-based ART methods outperformed their JD-based counterparts significantly. It is also found that there is a strong and significant positive correlation between CMD-based and JD-based ART methods. The correlation coefficients for the pairs (cmd-10, jd-10) and (cmd-flex, jd-flex) are .989 and .993, respectively ($p < .001$).

4) Comparing fixed-size and flexible-size candidate set ART algorithms in terms of F-measure: The fault-detection capabilities of the conventional FSCS-ART algorithm and Jiang et al.'s Flex-ART algorithm [21] are also compared. The only difference between the two algorithms is that FSCS-ART fixes the size of the candidate set to 10 whereas Flex-ART uses a candidate set of variable size depending on whether a new candidate can increase code coverage.

Out of the 28 faulty versions of *replace*, cmd-10 outperformed cmd-flex on 18 versions, with an average mean value 11.55% lower than that of the latter; jd-10 outperformed jd-flex on 17 versions, with an average mean value 5.00% lower than that of the latter. A paired-samples t test for cmd-10 and cmd-flex returned a p-value of .074$>$.05, and that for jd-10 and jd-flex returned a p-value of .284$>$.05, indicating that the differences of mean F-measures between the two algorithms were not statistically significant.

Relatively speaking, based on the above analysis, the algorithm for building the candidate set had a greater impact

on CMD-based ART than on JD-based ART.

It is also observed that there is a strong and significant positive correlation between the two algorithms. The correlation coefficients for the pairs (cmd-10, cmd-flex) and (jd-10, jd-flex) are .995 and .993, respectively ($p < .001$).

5) The APFD results: The APFD results for *replace* shown in Table II agree with the F-measure results, with cmd-10 being the highest (best), followed in descending order by cmd-flex, jd-10, jd-flex, *add* and RT. The differences between the mean APFD values, however, are very small. Even the worst one, RT, achieved a high score of 0.9600. This observation indicates that APFD may not necessarily be a suitable effectiveness measure or may need to be adjusted in certain situations, such as when the test suites are very large.

Despite the practically small differences in mean APFD values among the different methods, an ANOVA analysis (analysis of variance) over the 28 faulty versions successfully rejected the null hypothesis that the APFD means for the 6 techniques were equal at the 1% significance level. Further t tests for the pairs (cmd-10, cmd-flex), (cmd-10, jd-10), (cmd-10, jd-flex), (cmd-10, *add*) and (cmd-10, RT) also successfully rejected the null hypothesis at the 1% significance level, indicating that cmd-10 achieved (statistically significant) better APFD results than all the others for the *replace* program.

6) Summary: In summary, out of the 6 investigated methods for the *replace* program, cmd-10 was the best in terms of both F-measure and APFD. In the worst case, its F-measure was very close to that of RT.

C. Results of Experiments with the Space Program

1) Comparing with RT in terms of F-measure: Out of the 34 faulty versions of *space*, *add* outperformed RT on 31 versions and had an average mean F-measure 33.76% lower than RT. A paired-samples t test for RT and *add*, however, returned a p-value of .075>.05, indicating that the difference between the sample means of RT and *add* is not very significant. It is also found that the worst-case performance of *add* (as compared to RT) was poor: on version 27 of *space*, *add* used 661.00 test cases in average to detect the first failure; whereas RT used only 393.16 test cases in average.

For all of the 4 ART methods, their average mean F-measures on the *space* program are much lower than those of RT and *add*. The CMD-based ART methods cmd-10 and cmd-flex outperformed RT on 30 versions; the JD-based ART methods jd-10 and jd-flex, outperformed RT on 28 and 26 versions, respectively. Paired-samples t tests (over all of the 34 faulty versions of *space*) for (RT, cmd-10), (RT, cmd-flex), (RT, jd-10) and (RT, jd-flex) each returned a p-value of .001, indicating that all of the 4 ART methods outperformed RT significantly.

Among all of the 6 methods, cmd-10 had the best (lowest) average mean F-measure and the best (highest) mean APFD, and its worst-case performance (in terms of mean F-measure as compared to RT, out of 1,000 trials) was only about 10% higher than the F-measure of RT.

2) Comparing with the Additional algorithm in terms of F-measure: Out of the 34 faulty versions of *space*, *add* outperformed cmd-10, jd-10, cmd-flex and jd-flex on 26, 27, 27 and 28 versions, respectively. In this sense, *add* performed better than all of the ART methods. However, all these four ART methods outperformed *add* in terms of average mean F-measures greatly (see the fourth row of Table II). Paired-samples t tests for each pair of (*add*, ART method) returned a p-value greater than or equal to .196, indicating that the differences between the sample means of *add* and the ART methods are not statistically significant.

3) Comparing CMD and JD in terms of F-measure: Out of the 34 faulty versions of *space*, cmd-10 outperformed jd-10 on 29 versions, with an average mean value that is 20.66% ($= 1 - 29.68/37.41$) lower than that of jd-10; cmd-flex outperformed jd-flex on 27 versions, with an average mean value 0.88% lower than that of jd-flex. However, a paired-samples t test for cmd-10 and jd-10 returned a p-value of .073, and that for cmd-flex and jd-flex returned a p-value of .960, indicating that the differences between the sample means of CMD- and JD-based ART methods were not statistically significant.

It is also found that there is a significant positive correlation between the results of CMD-based and JD-based ART methods. The correlation coefficients for the pairs (cmd-10, jd-10) and (cmd-flex, jd-flex) are .885 and .771, respectively ($p < .001$).

4) Comparing fixed-size and flexible-size candidate set ART algorithms in terms of F-measure: The algorithm cmd-10 had an average mean F-measure that is 14.79% lower than that of cmd-flex. A paired-samples t test, however, returned a p-value of .348>.05, indicating that the difference is not statistically significant.

The algorithm jd-10 had an average mean F-measure that is 6.46% *higher* than that of jd-flex. A paired-samples t test returned a p-value of .005, indicating that jd-flex significantly outperformed jd-10.

In summary, for the *space* program, the algorithm for building the candidate set has had an impact on the effectiveness of ART: the fixed-size algorithm favored CMD in terms of reducing the average mean F-measure; whereas the flexible-size algorithm favored JD with greater statistical significance.

It is also observed that there is a strong and significant positive correlation between the two algorithms. The correlation coefficients for the pairs (cmd-10, cmd-flex) and (jd-10, jd-flex) are .830 and .999, respectively ($p < .001$).

5) The APFD results: As with the *replace* program, the differences between the mean APFD values for the

space program are very small. Even the worst one, RT, achieved a high score of 0.9931. Therefore, the differences in mean APFD scores do not have much practical significance, although an ANOVA analysis over the 34 faulty versions successfully rejected the null hypothesis that the APFD means for the 6 techniques were equal at the 1% significance level, and further t tests indicated that cmd-10 achieved (statistically significant) better APFD scores than every other method at the 1% significance level. The rankings of the APFD scores are consistent with the rankings of the average mean F-measures and are also consistent with the results of the *replace* program (with an exception of jd-10): the mean APFD of cmd-10 is the highest (best), followed in descending order by cmd-flex (0.99747), jd-flex (0.99745), *add*, jd-10 and RT.

6) Summary: In summary, the *space* results are consistent with the *replace* results: cmd-10 was the best among the 6 methods in terms of F-measure and APFD, and cmd-10 was also stable in that its worst-case performance (as compared to RT, in terms of mean F-measure) was close to the mean F-measure of RT.

V. Discussions and Conclusion

To answer the two research questions raised in Section I, a series of experiments have been conducted with two subject programs that have been widely used for software testing experiments in the literature, namely, the *replace* and *space* programs.

For the first research question, two frequency-based distance measures, namely, FMD and FHD, were proposed. Results of experiments using branch and statement coverage information show that the coverage-based distance measure (CMD) was superior to the frequency-based ones in revealing failures. In particular, branch CMD achieved the best results. Nevertheless, it was also found that frequency information is useful and can complement the coverage-based measure as FMD and FHD outperformed CMD greatly on several faulty versions of the subject programs. How to incorporate the frequency information into coverage-based measures will be a future research direction. We expect that an optimal strategy should combine both coverage and frequency information, with the former given a heavier weight.

To address the second research question, we investigated a group of 4 ART methods, which forms the Cartesian product of $\{\text{CMD, JD}\} \times \{\text{FSCS-ART, Flex-ART}\}$, together with the random and Additional algorithms that serve as experimental controls. Branch coverage information was used for test case prioritization; and both the F-measure and APFD were adopted as effectiveness measures. It has been found that CMD was superior to JD and that cmd-10 (that is, FSCS-ART using CMD) was the best among the 6 investigated algorithms on both the *replace* and the *space* programs.

In addition to answering the two research questions, the following findings have been reported in this paper: First, APFD may not necessarily always be a suitable measure for real-world very large test suites. It is found that, when test suites were large, even random testing achieved a high APFD score and the differences between different techniques became quite small – although such small differences still have a *statistical* significance, they do not have much *practical* significance. It is also found that there is a strong negative correlation between mean F-measure and mean APFD. That is, a lower mean F-measure would normally result in a higher mean APFD. The differences of mean F-measures between different techniques, however, are much greater than those of APFD.

A further finding is that the conventional Additional algorithm, which has been considered one of the best test case prioritization algorithms, can have a very poor performance for certain faulty programs. For example, its mean F-measure is 29 times as large as that of random testing (that is 2,628.99 vs 90.87) for version 15 of the *replace* program. This kind of observation was made for both the *replace* and *space* programs. In comparison, the ART algorithm cmd-10 was much more stable in that its worst-case mean F-measure was still close to the mean F-measure of random testing. Therefore, cmd-10 may be a more suitable choice for testers in situations where stability (or worst-case performance) of algorithms is critical. A further analysis shows that a reason for the poor performance of the Additional algorithm is that when a fault lies in a common execution path of the program code (that is, when a large number of test cases can reach the fault), trying to cover more branches/statements (as does the Additional algorithm) may not help to reveal a failure for that fault. In this situation, greater test case diversity as enforced by ART may be more effective than greater code coverage as enforced by the Additional algorithm. This observation suggests that knowledge about possible fault locations may help to select an effective test case prioritization technique.

For the internal validity of this research, the testing scripts and testing code written for this research have been carefully verified and the test results have also been carefully checked to ensure correctness.

There are, however, threats to external validity that concerns the generalization of the experimental results. In this research only two subject programs were studied with a small number of faults (28 for *replace* and 34 for *space*). The scale of study is therefore an obvious threat to external validity. Further empirical study with more subject programs will be required in the future.

References

[1] B. Hailpern and P. Santhanam, "Software debugging, testing, and verification," *IBM Systems Journal*, vol. 41, no. 1, pp. 4–12, 2002.

[2] Z. Q. Zhou, B. Scholz, and G. Denaro, "Automated software testing and analysis: Techniques, practices and tools," in *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. IEEE Computer Society Press, 2007.

[3] J. Burnim and K. Sen, "Heuristics for scalable dynamic test generation," in *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society Press, 2008, pp. 443–446.

[4] R. Hamlet, "Random testing," in *Encyclopedia of Software Engineering*, J. Marciniak, Ed. John Wiley & Sons, 2002, pp. 970–978.

[5] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou, "On favourable conditions for adaptive random testing," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 6, pp. 805–825, 2007.

[6] G. F. Renfer, "Automatic program testing," in *Proceedings of the 3rd Conference of the Computing and Data Processing Society of Canada*. University of Toronto Press, 1962.

[7] D. L. Bird and C. U. Munoz, "Automatic generation of random self-checking test cases," *IBM Systems Journal*, vol. 22, no. 3, pp. 229–245, 1983.

[8] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of Unix utilities," *Communications of the ACM*, vol. 33, no. 12, pp. 32–44, 1990.

[9] D. Slutz, "Massive stochastic testing of SQL," in *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98)*, 1998, pp. 618–622.

[10] J. E. Forrester and B. P. Miller, "An empirical study of the robustness of Windows NT applications using random testing," in *Proceedings of the 4th USENIX Windows Systems Symposium*, 2000, pp. 59–68.

[11] T. Dabóczi, I. Kollár, G. Simon, and T. Megyeri, "Automatic testing of graphical user interfaces," in *Proceedings of the 20th IEEE Instrumentation and Measurement Technology Conference (IMTC'03)*, 2003, pp. 441–445.

[12] T. Yoshikawa, K. Shimura, and T. Ozawa, "Random program generator for Java JIT compiler test system," in *Proceedings of the 3rd International Conference on Quality Software (QSIC'03)*. IEEE Computer Society Press, 2003, pp. 20–24.

[13] G. J. Myers, *The Art of Software Testing*. New York: Wiley, 1979.

[14] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive random testing: The ART of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.

[15] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Fault-based testing without the need of oracles," *Information and Software Technology*, vol. 45, no. 1, pp. 1–9, 2003.

[16] ——, "Semi-proving: An integrated method for program proving, testing, and debugging," *IEEE Transactions on Software Engineering*, vol. 37, no. 1, pp. 109–125, 2011.

[17] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Yang, "An innovative approach to tackling the boundary effect in adaptive random testing," in *Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*. IEEE Computer Society Press, 2007.

[18] I. K. Mak, "On the effectiveness of random testing," Master's thesis, The University of Melbourne, Melbourne, Australia, 1997.

[19] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "ARTOO: Adaptive random testing for object-oriented software," in *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, 2008.

[20] H. Jaygarl, C. K. Chang, and S. Kim, "Practical extensions of a randomized testing tool," in *Proceedings of the 33rd Annual International Computer Software and Applications Conference (COMPSAC'09)*. IEEE Computer Society Press, 2009, pp. 148–153.

[21] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, "Adaptive random test case prioritization," in *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE'09)*. IEEE Computer Society Press, November 2009, pp. 233–244.

[22] Z. Q. Zhou, "Using coverage information to guide test case selection in adaptive random testing," in *Proceedings of the 34th Annual International Computer Software and Applications Conference (COMPSAC'10), 7th International Workshop on Software Cybernetics*. IEEE Computer Society Press, 2010, pp. 208–213.

[23] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Proceedings of the 9th Asian Computing Science Conference (ASIAN'04), Lecture Notes in Computer Science 3321*. Springer-Verlag, 2004, pp. 320–329.

[24] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[25] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria," in *Proceedings of the 16th International Conference on Software Engineering (ICSE'94)*. IEEE Computer Society Press, 1994, pp. 191–200.

[26] C. Liu, X. Yan, and J. Han, "Mining control flow abnormality for logic error isolation," in *Proceedings of the 6th SIAM International Conference on Data Mining (SDM'06)*, 2006.

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

[28] Microsoft Corporation, "Jacek Czerwonka tester spotlight," http://msdn.microsoft.com/en-us/testing/cc136640.aspx (accessed in June 2011).

[29] SQLite homepage, http://www.sqlite.org (accessed in June 2011).