Is Operator-Based Mutant Selection Superior to Random Mutant Selection?

Lu Zhang[1,2], Shan-Shan Hou[1,2], Jun-Jue Hu[1,2], Tao Xie[3], Hong Mei[1,2]
[1]Institute of Software, School of Electronics Engineering and Computer Science
[2]Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
Peking University, Beijing, 100871, China
[3]Department of Computer Science, North Carolina State University, Raleigh, NC 27695
{zhanglu,houss,hujj08,meih}@sei.pku.edu.cn, xie@csc.ncsu.edu

ABSTRACT

Due to the expensiveness of compiling and executing a large number of mutants, it is usually necessary to select a subset of mutants to substitute the whole set of generated mutants in mutation testing and analysis. Most research on mutant selection focused on operator-based mutant selection, i.e., determining a set of sufficient mutation operators and selecting mutants generated with only this set of mutation operators. Recently, researchers began to leverage statistical analysis to determine sufficient mutation operators using execution information of mutants. However, whether mutants selected with these sophisticated techniques is superior to randomly selected mutants remains an open question. In this paper, we empirically investigate this open question by comparing three representative operator-based mutant-selection techniques with two random techniques. Our empirical results show that operator-based mutant selection is not superior to random mutant selection. These results also indicate that random mutant selection can be a better choice and mutant selection on the basis of individual mutants is worthy of further investigation.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Measurement, Experimentation

Keywords

Mutation testing, Test-adequacy criterion

1. INTRODUCTION

In software testing, test-adequacy criteria play an important role in determining whether the software under test (SUT) is adequately tested [16, 44]. With a test-adequacy criterion, a tester can continually create new test cases until the suite of test cases created so far satisfies the criterion. *Mutation testing*, which was first proposed by Hamlet [18] and DeMillo et al. [11], is an intensively studied way to construct such a test-adequacy criterion. In mutation testing, many faulty versions (known as *mutants*) of the SUT are generated through automatically changing the SUT with *mutation operators*, each of which is a rule to produce faulty versions and can be applied to various statements. Since the first proposal, mutation testing has attracted the attention of many researchers (e.g., Acree et al. [1], Budd et al. [7], Wong and Mathur [27, 41, 42], Offutt et al. [34, 30, 32, 33, 31, 26], Mresa et al. [29], Hierons and Harman [19], Barbosa et al. [4], and Zeller et al. [17, 37]). Recently, researchers used mutation operators to automatically produce faulty software to facilitate experimentation in research of software testing [6, 5, 28, 39]. Andrews et al. [3] and Do et al. [14] reported some evidence that faults generated with mutation operators are similar to real faults in evaluating test effectiveness. Following the terminology of Andrews et al. [3, 38], we refer to this way of using mutation as *mutation analysis*.

Mutation testing and mutation analysis are usually very expensive [7, 27, 41, 32, 38]. For example, Proteum [9] generates 4,937 mutants for *tcas* (which is the smallest program among the Siemens programs [21] and contains only 137 non-commenting and non-whitespace lines of code) using 108 mutation operators. Thus, compiling and executing a large number of mutants can be a big burden in mutation testing and analysis. To alleviate this burden, researchers have proposed various techniques [1, 27, 41, 42, 32, 4, 38] for selecting a subset of all the mutants. Naturally, researchers want the set of selected mutants to be similar to the set of all mutants. One simple technique is random mutant selection [1, 41, 42], in which, given a fixed percentage number (denoted as x), $x\%$ mutants are randomly selected. However, researchers seem to be more enthusiastic at investigating operator-based mutant selection [27, 41, 42, 32, 4, 38], which aims to select mutants generated with only a set of sufficient mutation operators[1]. While early research on operator-based mutant selection [41, 42, 32] tries to determine sufficient mutation operators via simple rules, recent research [4, 38] relies on sophisticated procedures to determine sufficient mutation operators involving statistical information that can be acquired only through executing a large number of mutants with a sufficiently large set of test cases.

Despite the enthusiasm in investigating operator-based mutant selection, whether operator-based mutant selection

[1]A set of mutation operators are sufficient if the mutants generated by the mutation operators can very much represent the mutants generated by all the mutation operators.

ICSE '10 May 2-8, 2010, Cape Town, South Africa

is superior to random mutant selection for mutation testing remains an open question. That is to say, despite recent progress in operator-based mutant selection (e.g., Offutt et al. [32], Barbosa et al. [4], and Siami Namin et al. [38]), there is lack of empirical evaluation of these operator-based mutant-selection techniques against random mutant selection techniques. Furthermore, as we can view random mutant selection as an approach to mutant selection with average effectiveness, answering the preceding open question can help us gain insights and deep understanding to the current research and achievements on mutant selection.

In this paper, we report an empirical study attempting to answer this open question. To evaluate the effectiveness of each mutant-selection technique, we adopt a widely used metric for evaluating mutant-selection techniques. The metric aims to measure to what extent the selected mutants are able to represent the whole set of mutants. Furthermore, we also evaluate the stability of each technique by checking how consistently the technique performs under different circumstances. For either effectiveness or stability, our empirical results do not support that operator-based mutant selection is superior to random mutant selection. That is to say, random mutant selection remains a competitive or even better choice compared with recent progress in operator-based mutant selection. Furthermore, as random mutant selection selects mutants on the basis of individual mutants instead of mutation operators, our empirical results also indicate that mutant-selection techniques based on individual mutants should be worthy of further investigation.

The main contributions of our study are as follows.

- Our study empirically evaluates three recent operator-based mutant-selection techniques (i.e., Offutt et al. [32], Barbosa et al. [4], and Siami Namin et al. [38]) against random mutant selection for mutation testing.
- Our study produces the first empirical results concerning stability of operator-based mutant selection and random mutant selection for mutation testing.
- Beside the random technique studied previously (referred to as the one-round random technique in this paper), our study also investigates another random technique involving two steps to select each mutant (referred to as the two-round random technique in this paper).
- The subjects used in our study are larger than those used in previous studies of random mutant selection. To the best of our knowledge, due to the extreme expensiveness of experimenting mutant-selection techniques, the Siemens programs are by far the largest subjects[2] used in studies of mutant selection [38].

We organize the rest of this paper as follows. Section 2 presents the experimental design in our study. Section 3 presents and analyzes the results obtained from our experiments. Section 4 discusses other issues in our study. Section 5 discusses related work. Section 6 concludes and presents some future work.

2. EXPERIMENTAL DESIGN

In this section, we first present the research questions in our study. Then, we describe the experimented techniques, the tool we used to obtain mutants, the subject programs, and the way we measure each technique. Finally, we describe the details of our experimental procedure.

2.1 Research Questions

In our study, we investigate the following research questions:

- **RQ1**: How does operator-based mutant selection compare with random mutant selection in terms of average effectiveness?
- **RQ2**: How does operator-based mutant selection compare with random mutant selection in terms of stability?

2.2 Experimented Techniques

In our study, we experimented three operator-based mutant-selection techniques (i.e., Offutt et al.'s 5 mutation operators [32], Barbosa et al.'s 10 mutation operators [4], and Siami Namin et al.'s 28 mutation operators [38])[3] and two random mutant-selection techniques.

Given a number (denoted as u), the first random mutant-selection technique is to randomly select u mutants. This technique is basically the x%-random technique studied by Wong and Mathur [41, 42]. The second random mutant-selection technique employs two steps when selecting each mutant. The first step randomly selects a mutation operator, and the second step randomly selects a mutant that is generated with the selected mutation operator. Using the two steps, the second random technique continually selects one mutant that has not been selected previously until u mutants are selected. In this paper, we refer to the first random technique as the *one-round random* and the second random technique as the *two-round random*. For the *one-round random*, the probability of selecting each mutant is equal; but for the *two-round random*, the number of selected mutants that are produced by each mutation operator is about the same.

2.3 Supporting Tool

In our study, we used Proteum [9], which is a mutation system implementing a comprehensive set of mutation operators for C programs, to generate mutants for each subject. The version of Proteum used in our study supports 108 mutation operators, including traditional mutation operators [2] and interface mutation operators [8]. As the 108 mutation operators include Offutt et al.'s five mutation operators[4], Barbosa et al.'s 10 mutation operators, and Siami Namin et al.'s 28 mutation operators, we are able to use Proteum to compare random mutant selection with all the three operator-based mutant-selection techniques.

2.4 Subject Programs

The subjects used in our study are the Siemens programs. The Siemens programs include seven C programs whose numbers of net lines of code (not counting whitespace and commenting lines) range from 137 to 513. Hutchins et al. [21]

[2]Some studies on mutation testing and analysis (such as Do et al. [14]) do use larger subjects, but they do not focus on mutant selection and they do not consider all the generated mutants.

[3]As Wong and Mathur's two mutation operators [41, 42] are among Offutt et al.'s five mutation operators [32] and Offutt et al. showed that any subset of the five mutation operators is not sufficient, we did not empirically compare Wong and Mathur's two mutation operators in our study.

[4]Offutt et al.'s five mutation operators are defined on programs in Fortran-77. Agrawal et al. [2] list the mutation operators in Proteum that correspond to Offutt et al.'s five mutation operators.

Table 1: Statistics of subjects

Program	Abb.	Net Lines of Code	Test Pool Size	All Mutants	Non-Equivalent Mutants
print_tokens	PT	343	4130	11741	9326
print_tokens2	PT2	355	4115	10266	8664
replace	RE	513	5542	23847	19861
schedule	SC	296	2650	4130	3670
schedule2	SC2	263	2710	6552	4832
tcas	TC	137	1608	4935	4069
tot_info	TI	281	1052	8767	7876

first introduced the Siemens programs in 1994, and after that many researchers (e.g., Rothermel et al. [35, 36], Elbaum et al. [15], Li et al. [24], Jones et al. [22], and Andrews et al. [3, 38]) used the Siemens programs as subjects in testing experiments. In particular, a recent study on mutant selection by Siami Namin et al. [38] used only the Siemens programs as subjects. For each of the Siemens programs, Hutchins et al. provided a test pool, and Rothermel et al. [35] augmented the test pool through manually adding some white-box test cases. After augmentation, the test pool for each program ensures that "each executable statement, edge, and definition-use pair in the base program or its control flow graph was exercised by at least 30 test cases" [35]. Table 1 depicts the statistics of the subjects. Note that the second column in Table 1 lists the abbreviations of the seven subjects, and we use these abbreviations to denote the subjects when presenting our experimental results in Section 3.

Similar to Siami Namin et al. [38], we considered the following three reasons when choosing our subjects. First, the Siemens programs contain typical structures that also appear in various large programs in C. Thus, findings on these subjects are very likely to generalize to other programs. Second, there is a large test pool for each of the Siemens programs. As measuring the effectiveness of selected mutants relies on the use of different test suites (see Section 2.5 for the details of measurement in our study), a large test pool allows us to construct a large number of test suites containing different test cases. Third, as Proteum generates a large number of mutants for even a small program, using programs significantly larger than the Siemens programs as subjects may result in huge computational cost. Actually, beside Siami Namin et al. [38], who used only less than one third of the mutants that Proteum generates for the Siemens programs[5], other researchers (i.e., Wong [41], Offutt et al. [32], and Barbosa et al. [4]) used programs much smaller than the Siemens programs for evaluating mutant-selection techniques.

2.5 Measurement

As our study is in the context of mutation testing, we adopted a metric that researchers used to evaluate the effectiveness of mutant-selection techniques in previous studies (e.g., Wong and Mathur [41, 42], Offutt et al. [32], and Barbosa et al. [4]). Given a program (denoted as P) and a set of mutants (denoted as AM) generated for P with all mutation operators, we removed equivalent mutants from AM and acquired a set of non-equivalent mutants (denoted as NEM). When evaluating a mutant-selection technique (denoted as T), we used T to select mutants from NEM, and denote the set of selected non-equivalent mutants as M_T. To evaluate the effectiveness of T, we create a series of test suites (denoted as $\{ts_1, ts_2, ..., ts_n\}$), each of which can kill all mutants in M_T. We denote the subset of mutants in NEM that can be killed by ts_i ($1 \leq i \leq n$) as $Killed_{NEM}(ts_i)$, and then we use Formula 1 to measure the effectiveness of T.

$$Eff(T) = \frac{\sum_{i=1}^{n} \frac{|Killed_{NEM}(ts_i)|}{|NEM|}}{n} \quad (1)$$

Intuitively, this metric measures the effectiveness of T as the representativeness of the set of non-equivalent mutants selected by T for the whole set of non-equivalent mutants NEM. As the aim of mutation testing is to provide a test-adequacy criterion, this metric measures the representativeness of M_T for NEM as the representativeness of the test-adequacy criterion based on M_T for the test-adequacy criterion based on NEM. Thus, the closer $Eff(T)$ is to 1.0, the more effective T is. When $Eff(T)$ is equal to 1.0, technique T is able to select a subset of mutants that fully represent the whole set of non-equivalent mutants.

As measuring the effectiveness of a subset of mutants in each experiment requires a series of test suites, we used a procedure similar to the procedure used by Offutt et al. [32] to construct the test suites. That is to say, for a subset of mutants, we continually selected k test cases from the test pool until the test suite composed of all the selected test cases is able to kill all the mutants in the subset. Offutt et al. [32] selected 200 (i.e., k=200) test cases each time when constructing such a test suite. That is to say, the numbers of test cases in test suites used by Offutt et al. are multiples of 200 (i.e., 200, 400, 600, etc.). Actually, Offutt et al. used this way of test-suite construction to simulate the situation of applying mutation testing as a test-adequacy criterion, and the number of test cases selected each time represents an increment of test cases in the process of building up each test suite for evaluating mutant selection. Considering that testers may use different incremental numbers to create the test suite, we used four different incremental numbers (i.e., k=25, 50, 100, and 200) including Offutt et al.'s incremental number. Note that one mutant-selection technique may achieve very different effectiveness values using test suites created with different incremental numbers. In our study, given an incremental number, we constructed 50 test suites when measuring the effectiveness of a subset of selected mutants.

2.6 Experimental Procedure

For each subject, we used all the 108 mutation operators in Proteum to generate mutants. The fifth column in Table 1 lists the number of all the generated mutants for each subject.

After acquiring all the mutants, for each subject, we executed each test case in the test pool of the subject against each mutant of the subject and the subject in the original form. Thus, we acquired the information of which mutants are killed by which test cases for each subject. Similar to Siami Namin et al. [38], we deemed mutants that cannot be killed by any test case as equivalent mutants in our study. The last column in Table 1 lists the number of non-equivalent mutants for each subject.

For a subject, different operator-based mutant-selection techniques select different numbers of mutants. Thus, it is difficult for us to compare random mutant selection with

[5]Except for the smallest subject (i.e., *tcas*), Siami Namin et al. [38] used 2000 mutants for each other subject.

Table 2: Offutt et al.'s technique v.s. random mutant selection

Incr	Program		PT		PT2		RE		SC		SC2		TC		TI	
	Result		Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev
25	Offutt et al.		99.11	0.27	99.84	0.17	99.09	0.29	99.94	0.07	99.29	0.23	99.54	0.21	99.57	0.32
	one-round random	50%	99.09	0.21	99.52	0.14	99.20	0.15	99.11	0.38	99.09	0.29	98.16	0.49	99.76	0.11
		75%	99.35	0.16	99.74	0.10	99.48	0.09	99.44	0.26	99.21	0.25	98.56	0.37	99.84	0.07
		100%	99.52	0.12	99.79	0.08	99.57	0.07	99.58	0.18	99.44	0.17	98.81	0.30	99.87	0.05
	two-round random	50%	98.60	0.28	99.40	0.16	99.04	0.19	99.38	0.24	99.03	0.30	98.15	0.59	99.67	0.15
		75%	99.02	0.22	99.58	0.12	99.30	0.14	99.50	0.20	99.32	0.23	98.66	0.34	99.77	0.11
		100%	99.18	0.19	99.70	0.10	99.40	0.11	99.59	0.17	99.52	0.18	98.86	0.30	99.80	0.10
50	Offutt et al.		99.26	0.21	99.91	0.13	99.27	0.20	99.97	0.04	99.40	0.23	99.68	0.11	99.66	0.17
	one-round random	50%	99.14	0.21	99.58	0.12	99.42	0.12	99.31	0.34	99.17	0.27	98.62	0.44	99.79	0.11
		75%	99.47	0.15	99.79	0.08	99.57	0.08	99.53	0.25	99.32	0.23	98.95	0.33	99.88	0.06
		100%	99.60	0.12	99.89	0.06	99.61	0.08	99.62	0.20	99.58	0.15	99.13	0.25	99.90	0.06
	two-round random	50%	98.68	0.30	99.48	0.14	99.22	0.16	99.48	0.23	99.32	0.25	98.60	0.50	99.73	0.14
		75%	99.18	0.22	99.67	0.10	99.37	0.14	99.64	0.17	99.46	0.20	98.95	0.32	99.80	0.11
		100%	99.28	0.21	99.77	0.08	99.53	0.09	99.68	0.14	99.65	0.14	99.18	0.25	99.90	0.06
100	Offutt et al.		99.34	0.23	99.97	0.05	99.54	0.18	99.98	0.02	99.62	0.21	99.80	0.13	99.74	0.20
	one-round random	50%	99.32	0.20	99.66	0.11	99.53	0.10	99.43	0.34	99.36	0.25	98.99	0.38	99.83	0.10
		75%	99.56	0.15	99.77	0.08	99.65	0.08	99.65	0.21	99.52	0.19	99.23	0.28	99.92	0.05
		100%	99.62	0.13	99.93	0.04	99.71	0.07	99.73	0.15	99.68	0.14	99.37	0.23	99.94	0.04
	two-round random	50%	99.00	0.27	99.46	0.15	99.36	0.15	99.60	0.20	99.44	0.23	99.05	0.38	99.74	0.17
		75%	99.20	0.23	99.59	0.12	99.50	0.12	99.70	0.16	99.69	0.15	99.32	0.25	99.88	0.08
		100%	99.46	0.19	99.76	0.08	99.61	0.10	99.75	0.12	99.72	0.13	99.45	0.21	99.90	0.06
200	Offutt et al.		99.54	0.26	99.97	0.07	99.65	0.17	99.99	0.01	99.60	0.18	99.89	0.11	99.76	0.20
	one-round random	50%	99.46	0.20	99.72	0.08	99.63	0.11	99.64	0.25	99.54	0.22	99.26	0.35	99.93	0.06
		75%	99.59	0.17	99.83	0.07	99.73	0.09	99.75	0.16	99.62	0.17	99.47	0.27	99.94	0.05
		100%	99.79	0.09	99.92	0.04	99.83	0.06	99.78	0.13	99.75	0.13	99.60	0.21	99.95	0.04
	two-round random	50%	99.15	0.28	99.61	0.12	99.48	0.14	99.70	0.18	99.65	0.18	99.31	0.33	99.84	0.12
		75%	99.35	0.25	99.73	0.09	99.65	0.10	99.77	0.14	99.78	0.12	99.54	0.23	99.89	0.08
		100%	99.57	0.19	99.85	0.06	99.71	0.09	99.82	0.10	99.81	0.11	99.63	0.20	99.93	0.05

all the three techniques on the same ground. Therefore, we used the following way to compare an operator-based mutant-selection technique with random mutant-selection.

When comparing an operator-based mutant-selection technique (denoted as T) with random mutant selection on one subject, we used T to select a subset of mutants (denoted as M_T) from all the non-equivalent mutants (denoted as NEM) of the subject. To compare T with random mutant selection, we used each random mutant-selection technique to select a series of subsets of mutants from NEM, each subset containing $50\% * |M_T|$, $75\% * |M_T|$, and $100\% * |M_T|$ mutants. To reduce accidental results, for each random technique and each size of subsets, we randomly selected m subsets of the same size. That is to say, for each subject and each random technique, we randomly selected m subsets each containing $50\% * |M_T|$ mutants, m subsets each containing $75\% * |M_T|$ mutants, and m subsets each containing $100\% * |M_T|$ mutants. After acquiring the subsets of mutants selected with T and the random mutant-selection techniques, we used the metric defined in Section 2.5 to measure the effectiveness of each technique. For each random technique and each size of subsets (e.g., using a random technique to select $100\% * |M_T|$ mutants), we used the average effectiveness of the m subsets as the effectiveness of that technique with that size. In our study, we set the value of m as 50, which is large enough to avoid accidental results.

As we used 50 test suites to measure the effectiveness of each technique and we randomly selected 50 subsets of mutants for each random technique, we further studied the stability of each technique in terms of standard deviation of its effectiveness.

3. RESULTS AND ANALYSIS

In this section, we present and analyze the results of comparison to answer the two research questions. We further analyze the ability of reduction in mutants for the three operator-based mutant-selection techniques. The detailed results of our evaluation are available at `https://sites.google.com/site/asergrp/projects/ranmu`.

3.1 Effectiveness

Tables 2, 3, and 4 depict the average effectiveness values and standard-deviation values for comparing Offutt et al.'s technique, Barbosa et al.'s technique, and Siami Namin et al.'s technique with random mutant selection, respectively[6]. In the three tables, we use $Incr$ to represent the four incremental numbers for creating test suites for evaluating selected mutants; Eff to represent average effectiveness values measured by the metric defined in Section 2.5; Dev to represent standard-deviation values among the corresponding effectiveness values; and 50%, 75%, and 100% to represent the use of random mutant selection to select 50%, 75%, and 100% of mutants selected by each of the three operator-based mutant-selection techniques. Both the effectiveness values and the standard-deviation values are in percentage

[6]Due to space limit, we use tables instead of figures to present the experimental results. Tables are more concise but less intuitive than figures. Also due to space limit, we present results on effectiveness and standard deviation in each table.

Table 3: Barbosa et al.'s technique v.s. random mutant selection

Incr	Program		PT		PT2		RE		SC		SC2		TC		TI	
	Result		Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev
25	Barbosa et al.		99.20	0.21	99.89	0.13	99.42	0.18	99.97	0.02	99.73	0.13	99.57	0.13	99.62	0.23
	one-round random	50%	99.61	0.10	99.91	0.04	99.64	0.06	99.50	0.21	99.45	0.18	98.93	0.26	99.85	0.07
		75%	99.73	0.07	99.93	0.03	99.75	0.05	99.60	0.18	99.61	0.14	99.26	0.19	99.91	0.04
		100%	99.80	0.05	99.94	0.03	99.81	0.04	99.76	0.11	99.73	0.10	99.45	0.15	99.92	0.04
	two-round random	50%	99.39	0.16	99.86	0.06	99.54	0.09	99.59	0.16	99.46	0.19	99.01	0.25	99.79	0.10
		75%	99.65	0.11	99.93	0.03	99.66	0.05	99.70	0.11	99.68	0.13	99.30	0.18	99.85	0.07
		100%	99.78	0.08	99.94	0.03	99.75	0.05	99.76	0.09	99.77	0.09	99.47	0.15	99.91	0.04
50	Barbosa et al.		99.31	0.23	99.96	0.05	99.60	0.19	99.98	0.02	99.82	0.11	99.70	0.14	99.73	0.20
	one-round random	50%	99.68	0.10	99.94	0.04	99.71	0.06	99.63	0.18	99.46	0.19	99.29	0.22	99.88	0.06
		75%	99.78	0.07	99.96	0.02	99.80	0.04	99.74	0.13	99.72	0.11	99.47	0.17	99.92	0.04
		100%	99.83	0.05	99.97	0.02	99.85	0.04	99.75	0.11	99.78	0.09	99.61	0.14	99.95	0.03
	two-round random	50%	99.49	0.17	99.89	0.04	99.57	0.08	99.67	0.15	99.61	0.16	99.27	0.22	99.85	0.08
		75%	99.69	0.10	99.96	0.02	99.73	0.05	99.74	0.12	99.74	0.12	99.53	0.16	99.91	0.05
		100%	99.82	0.08	99.97	0.02	99.79	0.04	99.80	0.09	99.82	0.08	99.67	0.13	99.93	0.04
100	Barbosa et al.		99.45	0.21	99.96	0.07	99.66	0.17	99.99	0.02	99.84	0.12	99.84	0.11	99.77	0.16
	one-round random	50%	99.77	0.09	99.96	0.02	99.78	0.06	99.69	0.17	99.62	0.16	99.48	0.20	99.92	0.05
		75%	99.86	0.06	99.97	0.02	99.84	0.04	99.77	0.13	99.77	0.11	99.63	0.16	99.94	0.04
		100%	99.88	0.04	99.97	0.02	99.89	0.03	99.84	0.09	99.84	0.09	99.78	0.11	99.96	0.03
	two-round random	50%	99.61	0.16	99.94	0.03	99.70	0.07	99.74	0.12	99.65	0.16	99.54	0.19	99.89	0.07
		75%	99.79	0.09	99.96	0.02	99.79	0.05	99.78	0.10	99.83	0.09	99.71	0.14	99.91	0.06
		100%	99.87	0.06	99.98	0.02	99.84	0.04	99.84	0.08	99.89	0.06	99.80	0.11	99.94	0.04
200	Barbosa et al.		99.61	0.24	99.99	0.01	99.80	0.15	99.99	0.01	99.90	0.06	99.89	0.11	99.79	0.20
	one-round random	50%	99.82	0.09	99.96	0.02	99.83	0.06	99.78	0.14	99.70	0.15	99.63	0.20	99.95	0.04
		75%	99.89	0.05	99.99	0.01	99.89	0.04	99.83	0.09	99.87	0.08	99.82	0.13	99.96	0.03
		100%	99.92	0.04	99.98	0.01	99.92	0.03	99.89	0.07	99.87	0.08	99.87	0.10	99.97	0.03
	two-round random	50%	99.71	0.15	99.95	0.03	99.77	0.08	99.81	0.11	99.81	0.11	99.67	0.18	99.90	0.07
		75%	99.87	0.07	99.99	0.01	99.85	0.05	99.86	0.08	99.90	0.07	99.82	0.13	99.95	0.04
		100%	99.92	0.05	99.98	0.01	99.89	0.04	99.88	0.08	99.93	0.05	99.88	0.10	99.96	0.03

points. To make the difference clear, we keep two digits after the decimal point for each value. From the three tables, we have the following observations related to the average effectiveness.

First, our results confirm that all the three operator-based mutant-selection techniques are able to achieve good effectiveness values using test suites created with all the four incremental numbers. According to the criterion proposed by Offutt et al. [32], a mutant-selection technique is good in effectiveness if it achieves an average effectiveness value over 99% using test suites created with incremental number 200. Based on this criterion, all the three technique are good in effectiveness, except for Siami Namin et al.'s technique on *tcas* (i.e., 98.79%).

Second, despite the effectiveness of the three operator-based mutant-selection techniques, none of them is superior to random mutant-selection when selecting the same number of mutants. Both random techniques outperform Offutt et al.'s technique for four subjects (i.e., *print_tokens*, *replace*, *schedule*2, and *tot_info*) out of seven. This trend is consistent for all the four incremental numbers. Both random techniques consistently outperform Barbosa et al.'s technique for *print_tokens*, *replace*, and *tot_info* with different incremental numbers. For *print_tokens*2 and *schedule*2, the two random techniques achieve almost the same average effectiveness as Barbosa et al.'s technique. Both random techniques consistently outperform Siami Namin et al.'s technique for *replace*, *tcas*, and *tot_info* with different incremental numbers. The two-round random technique also achieves similar effectiveness values as Siami Namin et al.'s technique for *schedule* for different incremental numbers. Overall, for any subject, the difference between one operator-based mutant-selection technique and its corresponding random mutant-selection techniques is quite small. This observation indicates that random mutant selection is still as competitive as or even better than operator-based mutant-selection in terms of average effectiveness.

Third, compared with the three operator-based mutant-selection techniques, random mutant selection is able to achieve competitive effectiveness when selecting fewer mutants. In general, for any random technique and any subject, the differences between selecting 50%, 75%, and 100% mutants are quite small. Thus, the difference between each operator-based mutant-selection technique and its corresponding random techniques selecting 50% or 75% mutants are also small. Based on Offutt et al.'s criterion, for each operator-based mutant-selection technique, using its corresponding random techniques to select 50% is also good in effectiveness, since the average effectiveness values are always over 99% for all the subjects using test suites created with incremental number 200. Since there is no sophisticated strategy in random mutant selection, this observation indicates that it is likely that mutants selected by either of the three operator-based mutant-selection techniques are more than necessary.

Fourth, when comparing the two random techniques, the one-round technique is less effective than the two-round technique for smaller subjects, but more effective than the two-round technique for larger subjects. We suspect the reason to be that the one-round technique may fail to select any

Table 4: Siami Namin et al.'s technique v.s. random mutant selection

Incr	Program		PT		PT2		RE		SC		SC2		TC		TI	
	Result		Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev	Eff	Dev
25	Siami Namin et al.		99.71	0.14	99.95	0.02	99.36	0.11	99.60	0.13	99.60	0.19	97.58	0.54	98.94	0.32
	one-round random	50%	99.30	0.17	99.70	0.10	99.24	0.14	99.11	0.37	98.99	0.30	98.23	0.48	99.68	0.14
		75%	99.47	0.13	99.80	0.07	99.42	0.12	99.40	0.26	99.17	0.29	98.60	0.38	99.75	0.12
		100%	99.57	0.10	99.89	0.05	99.56	0.08	99.49	0.21	99.43	0.18	98.94	0.27	99.85	0.07
	two-round random	50%	98.71	0.26	99.61	0.11	99.01	0.18	99.20	0.31	99.08	0.26	98.30	0.46	99.57	0.19
		75%	99.13	0.21	99.67	0.10	99.26	0.14	99.49	0.20	99.34	0.21	98.70	0.32	99.71	0.13
		100%	99.38	0.16	99.85	0.06	99.37	0.11	99.59	0.16	99.51	0.18	98.97	0.27	99.72	0.14
50	Siami Namin et al.		99.81	0.10	99.97	0.02	99.47	0.11	99.66	0.13	99.64	0.19	98.08	0.5	99.16	0.42
	one-round random	50%	99.31	0.17	99.83	0.08	99.38	0.12	99.21	0.41	99.13	0.28	98.72	0.42	99.81	0.09
		75%	99.53	0.13	99.85	0.06	99.55	0.09	99.48	0.25	99.42	0.21	98.99	0.31	99.80	0.10
		100%	99.64	0.11	99.91	0.04	99.66	0.07	99.57	0.22	99.52	0.16	99.14	0.25	99.86	0.08
	two-round random	50%	99.03	0.23	99.62	0.11	99.22	0.16	99.40	0.29	99.33	0.25	98.79	0.39	99.64	0.18
		75%	99.32	0.20	99.74	0.09	99.37	0.13	99.60	0.18	99.43	0.21	99.07	0.29	99.69	0.17
		100%	99.51	0.15	99.87	0.05	99.50	0.11	99.64	0.16	99.63	0.15	99.28	0.22	99.79	0.11
100	Siami Namin et al.		99.87	0.10	99.98	0.02	99.61	0.08	99.73	0.07	99.68	0.20	98.56	0.51	99.25	0.40
	one-round random	50%	99.52	0.16	99.78	0.08	99.47	0.12	99.46	0.33	99.31	0.27	99.12	0.33	99.79	0.13
		75%	99.63	0.13	99.92	0.04	99.64	0.08	99.61	0.23	99.54	0.18	99.36	0.24	99.86	0.09
		100%	99.76	0.09	99.95	0.03	99.73	0.06	99.67	0.19	99.65	0.16	99.45	0.21	99.91	0.06
	two-round random	50%	99.16	0.25	99.64	0.11	99.32	0.15	99.57	0.23	99.43	0.22	99.12	0.32	99.64	0.20
		75%	99.43	0.19	99.83	0.07	99.54	0.11	99.68	0.16	99.62	0.17	99.33	0.25	99.78	0.14
		100%	99.61	0.17	99.90	0.05	99.64	0.09	99.75	0.13	99.71	0.14	99.49	0.20	99.83	0.10
200	Siami Namin et al.		99.94	0.04	99.99	0.01	99.67	0.07	99.75	0.10	99.79	0.18	98.79	0.44	99.44	0.37
	one-round random	50%	99.59	0.17	99.83	0.07	99.20	0.11	99.61	0.27	99.58	0.21	99.39	0.29	99.86	0.10
		75%	99.76	0.11	99.96	0.02	99.74	0.08	99.74	0.18	99.68	0.16	99.52	0.23	99.92	0.06
		100%	99.80	0.09	99.96	0.02	99.80	0.07	99.78	0.14	99.73	0.14	99.60	0.21	99.91	0.06
	two-round random	50%	99.33	0.24	99.77	0.09	99.50	0.14	99.68	0.20	99.60	0.19	99.33	0.33	99.78	0.16
		75%	99.57	0.17	99.85	0.05	99.64	0.11	99.75	0.15	99.72	0.14	99.53	0.24	99.85	0.10
		100%	99.70	0.15	99.94	0.03	99.71	0.09	99.81	0.11	99.80	0.12	99.69	0.18	99.85	0.11

mutant for some important mutation operators when the number of mutants is small. Without mutants generated with these mutation operators, the selected set of mutants is not as representative as those containing mutants generated with all the different mutation operators. However, for the two-round technique, the distribution of selected mutants over the mutation operators may be very different from the distribution of all the non-equivalent mutants over the mutation operators when the subject becomes large. Thus, mutants selected by the two-round technique may be less representative than those selected by the one-round technique for large subjects.

Finally, although different incremental numbers impact the effectiveness values for any experimented techniques and any subjects, the preceding observations are consistent for different incremental numbers. Typically, with the increase of the incremental number, the effectiveness value for any technique on any subject also increases. We suspect the reason to be that different incremental numbers result in different sizes of created test suites and the differences in test-suite sizes lead to the differences in effectiveness values. We further checked the average sizes of test suites under different incremental numbers and corroborated this suspicion. Table 5 depicts average test-suite sizes for measuring Offutt et al.'s technique on the seven subjects under different incremental numbers. The trends in average test-suite sizes for other experimented techniques are similar. Due to space limit, we do not present the average test-suite sizes for them here. It should also be noted that, due to the nature of the metric used in our study, the test suites created with any incremental number usually are of different sizes, and thus the average test-suite sizes are not multiples of 25, 50, 100, or 200.

Table 5: Average test-suite sizes for Offutt et al.'s

Program	Increment 25	Increment 50	Increment 100	Increment 200
PT	309	479	725	1190
PT2	196	307	471	725
RE	638	1019	1609	2445
SC	237	383	591	894
SC2	290	464	710	970
TC	474	700	910	1175
TI	185	261	351	468

3.2 Stability

Based on the standard-deviation values depicted in Tables 2, 3, and 4, we have the following observations on comparing the stability of the three operator-based mutant-selection techniques and random mutant selection techniques.

First, all the experimented techniques are quite stable in effectiveness. For any experimented technique, the standard-deviation values are typically less than 0.20 percentage points, and standard-deviation values are rarely larger than 0.30 percentage points. Most of those exceptionally large standard-deviation values come from Siami Namin et al.'s technique and random techniques with 50% mutants on *tcas*, which is the smallest subject. Siami Namin et al.'s technique is less effective and less stable for *tcas*. Random techniques seem to be less stable for smaller subjects but more stable

Table 6: Ratio of selected mutants

Program	Offutt et al.	Barbosa et al.	Siami Namin et al.
PT	6.00	14.89	7.33
PT2	5.90	18.89	8.64
RE	6.84	16.30	6.62
SC	7.77	14.11	7.11
SC2	8.09	14.84	7.88
TC	6.07	15.31	7.08
TI	10.27	17.92	7.29
Average	7.28	16.04	7.42

for larger subjects.

Second, similar to the observation on average effectiveness, either of the three operator-based mutant-selection techniques is not more stable than random mutant selection when selecting the same number of mutants. Both random techniques are more stable than Offutt et al.'s technique for *print_tokens*, *replace*, *schedule*2, and *tot_info*; and are as stable as Offutt et al.'s technique for *print_tokens*2. Both random techniques are more stable than Barbosa et al.'s technique for *print_tokens*, *replace*, and *tot_info*; and are as stable as Barbosa et al.'s technique for *print_tokens*2, *schedule*2, and *tcas*. Both random techniques are more stable than Siami Namin et al.'s technique for *schedule*2, *tcas*, and *tot_info*; and are as stable as Siami Namin et al.'s technique for *replace*. It is interesting to note that a technique achieving better effectiveness values is typically also more stable.

Third, when comparing the two random techniques, the one-round technique seems to be less stable than the two-round technique for smaller subjects, but more effective than the two-round technique for larger subjects. This observation is in accordance with the observation on comparing the effectiveness of the two random techniques. Furthermore, given a random mutant-selection technique, selecting more mutants is more stable than selecting fewer mutants.

3.3 Reduction in Mutants

As the preceding results indicate that none of the three operator-based mutant-selection techniques is superior to random mutant selection in terms of either effectiveness or stability, we further examine their ability to reduce the number of mutants. Table 6 depicts the ratio of selected mutants to all the non-equivalent mutants for each subject. From this table, we have the following observations.

First, all the three operator-based mutant-selection techniques are able to substantially reduce the number of mutants. Either Offutt et al.'s technique or Siami Namin et al.'s technique is able to reduce about 93% mutants, while Barbosa et al.'s technique is able to reduce 84% mutants. Our result for Siami Namin et al.'s technique is similar to that reported by Siami Namin et al. [38] (i.e., 92.6%). Our result for Offutt et al.'s technique is different from that reported by Offutt et al. [32] (i.e., 78%). We suspect the reason to be that we considered much more mutation operators than Offutt et al. Our result for Barbosa et al.'s technique is also different from that reported by Barbosa et al. [4] (i.e., 78%). We suspect the reason to be that Barbosa et al. used 71 mutation operators (which were implemented in Proteum by 2001) rather than 108 mutation operators in our study. This observation also indicates that Barbosa et al.'s technique is much more expensive than either Offutt et al.'s technique or Siami Namin et al.'s technique.

Second, for each technique, the ratio of selected mutants does not differ much for a different subject. This observation indicates that the reduction in mutants is highly predictable for any of three operator-based mutant-selection techniques. Furthermore, when applying random mutant selection in practice, we may use the average ratio of selected mutants for an operator-based mutant-selection technique as a guidance to determine the number of mutants to select. For example, randomly selecting 7% mutants from mutants generated with the 108 Proteum mutation operators is likely to achieve similar effectiveness and stability as Offutt et al.'s technique.

4. DISCUSSION

In this section, we discuss the following issues related to our study: threats to validity, cost of mutant selection, and some further implications of our experimental results.

4.1 Threats to Validity

4.1.1 Internal Validity

Threats to internal validity are concerned with uncontrolled factors that may also be responsible for the results. In our study, the main threat to internal validity is the possible faults in our experiments and result analysis. To reduce this threat, we used Proteum for mutant generation. Furthermore, we reviewed all the code that we produced for our experiments and analysis before conducting the experiments. Note that the experimented techniques are all straightforward to implement and their implementation is thus less likely to threaten the internal validity.

4.1.2 External Validity

Threats to external validity are concerned with whether the findings in our study are generalizable for other situations. The main threat to external validity lies in the representativeness of the subjects. To reduce this threat, we chose seven subjects written in C and these subjects contain a wide range of data and control structures commonly used in C (or even C++ and Java) programs [38]. Conducting more experiments using more subjects of larger sizes and with structures not contained in our subjects is one way to further reduce this threat. Note that, when experimenting with subjects containing object-oriented structures, mutation operators on these structures [26] should also be considered.

4.1.3 Construct Validity

Threats to construct validity are concerned with whether the measurement in our study reflects real-world situations. The main threat to construct validity is the way we measured the effectiveness of selected mutants. To reduce this threat, we used a metric commonly used by previous studies, such as Offutt et al. [32] and Barbosa et al. [4]. Further reduction of this threat requires the design of better metrics to assess the effectiveness of selected mutants in mutation testing. The metric used in our study actually measures to what extent the selected mutants are representative for mutants generated with all the mutation operators. Indeed, users of mutant-selection techniques may be more concerned with the representativeness of the selected mutants for real faults. Such concerns may be alleviated by previous empirical studies [3, 14] showing evidence that mutants generated with mutation operators are similar to real faults in evaluating test effectiveness. But it is worthwhile of conducting

empirical studies directly on representativeness of the selected mutants for real faults in future work. Furthermore, the metric used in our study requires a series of randomly created test suites, but test suites used in practice may not be created randomly.

4.2 Cost of Mutant Selection

According to Offutt et al. [32], the expensiveness of mutation testing mainly lies in the need to compile and execute a large number of mutants against each test case. Compared with the cost of compiling and executing mutants, the cost of mutant selection is typically very small. In our study, we used Proteum as the platform for mutant selection. In particular, we ran Proteum on a PC with a Genuine Intel CPU T1400 at 1.83GHz and 1GB memory running SUSE Linux (version 2.6.25.5-1.1) with the *tcsh* shell (version 6.15.00). For each subject, Proteum generated the selected mutants in a few seconds. For the smallest subject (i.e., *tcas*), the time is less than one second; for the largest subject (i.e., *replace*), the time is less than four seconds; for each other subject, the time is less than two seconds. But the total time for us to compile and execute all the mutants for all the seven subjects against all the test cases under the same hardware and software environment is about one month CPU time. Note that Offutt et al.'s, Barbosa et al.'s, and Siami Namin et al.'s techniques averagely select 7.28%, 16.04%, and 7.42% mutants, respectively. That is to say, compiling and executing mutants selected by one of the three operator-based mutant-selection techniques for all the seven subjects against all the test cases may take two to five days. As a result, for either operator-based mutant selection or random mutant selection, there is little difference in the cost of mutant selection.

One issue that we may need to consider is the way that Barbosa et al.'s and Siami Namin et al.'s techniques use to determine mutant operators. In particular, both techniques determine the set of sufficient mutation operators using training data acquired through compiling and executing mutants against test cases. That is to say, both techniques may require some extra cost besides compiling and executing the selected mutants. However, as their cross-validation indicates that the set of sufficient mutation operators determined with some programs may also be applicable for other programs, the extra cost of either Barbosa et al.'s technique or Siami Namin et al.'s technique should be considerably small.

4.3 Further Implications

Based on the findings of our study, we have the following implications.

First, as the three operator-based mutant-selection techniques are not superior to the two random mutant-selection techniques in terms of either effectiveness or stability, random mutant-selection techniques may be a better choice in practice due to their flexibility in controlling the number of selected mutants. Note that random mutant-selection techniques can achieve similar effectiveness and stability even when selecting much fewer mutants.

Second, the findings in our study also imply that we may need much fewer mutants in mutation testing than those selected by existing operator-based mutant-selection techniques. That is to say, it is very likely to invent new mutant-selection techniques to select much fewer mutants but with similar or even better effectiveness and stability. Random mutant selection may be a good starting point.

Third, considering the nature of random mutant selection, the following difference between random mutant selection and operator-based mutant selection may be an explanation of the surprisingly good results of random mutant selection. Random mutant selection selects mutants on the basis of individual mutants, but operator-based mutant selection needs to include or exclude all the mutants generated with one mutation operator as a whole. If this difference accounts for the goodness of random mutant selection, techniques that both consider the difference in operators and select mutants individually may outperform existing random and operator-based mutant-selection techniques.

5. RELATED WORK

We next discuss related work on mutation testing and analysis, and mutant selection.

5.1 Mutation Testing and Analysis

Mutation testing is a fault-based testing approach, which is first proposed by Hamlet [18] and DeMillo et al. [11]. In mutation testing, the primary aim is to provide a rigorous test-adequacy criterion that can help enhance test suites. For the software under test (SUT), a tester can use mutation operators to generate a number of mutants. After identifying equivalent mutants, the tester can use the remaining non-equivalent mutants to enhance test suites. That is to say, if a test suite cannot kill all the remaining non-equivalent mutants, more test cases may be required to enhance the test suite. Another usage of mutation is mutation analysis, whose aim is not to enhance test suites but to provide assessment of test effectiveness to facilitate experiments in testing research. It is interesting to note that researchers such as Briand et al. [6] have already used mutation faults to measure test effectiveness even before the empirical confirmation of its appropriateness by Andrews et al. [3] and Do et al. [14].

For both mutation testing and analysis, a major concern is the expensiveness of compiling and executing a large number of mutants. In the literature, there are mainly four categories of techniques to reduce this cost. The first category is to select a subset of mutants as substitute. As our research in this paper falls into this category, we detailedly discuss research in this category in Section 5.2. The second category is to use low-cost ways to determine which test case kills which mutant. Weak mutation [20] and firm mutation [43] are two representative techniques in this category. The third category is to use efficient ways to generate, compile and execute mutants. As mutants only slightly differ from the original program, taking the advantage of the commonalities of the mutants may accelerate the generation, compilation and execution of the mutants. Compiler-integrated mutation [10] and schema-based mutation [40] are two representative techniques in this category. The last category is to compile and execute mutants in parallel. Researchers have investigated parallel compilation and execution of mutants on different computer architectures [23, 34]. Techniques in different categories are typically complementary to each other, as they can be combined together to reduce the cost of mutant compilation and execution.

The second concern in mutation testing and analysis is the cost in identifying equivalent mutants. In the literature, researchers proposed several techniques for detecting equivalent mutants statically [30, 33, 19] or dynamically [17, 37].

For mutation testing, there is still another concern: the cost of acquiring mutation-adequate test suites. In the literature, researchers also proposed some techniques to automatically generate mutation-adequate test suites [13, 31, 25]. Techniques addressing both concerns are also complementary to research on mutant selection. As our research in this paper mainly considers mutant selection, further discussion related to these two concerns is out of the scope of this paper.

5.2 Mutant Selection

As mentioned previously, mutant selection is an important way to reduce the cost of mutation testing and analysis. Since Mathur [27] first proposed the idea of excluding some mutation operators in mutation testing, several researchers have studied operator-based mutant selection. In operator-based mutant selection, researchers first determine a set of mutation operators, and then select only mutants generated with this set of mutation operators.

Wong and Mathur [41, 42] studied 2 mutation operators among the 22 mutation operators in Mothra [12], and found that mutants generated with these 2 mutation operators can achieve similar results as mutants generated with all the 22 mutation operators. Offutt et al. [32] experimentally determined 5 mutation operators among the 22 mutation operators in Mothra, and found that the effectiveness values of the 5 mutation operators are between 99.0% and 100% on ten subjects, with the average 99.5%. Note that the effectiveness values are based on test suites created with incremental number 200. Furthermore, Offutt et al. also found that, without any of the 5 mutation operators, the effectiveness value on some subject would be lower than 99%, which they defined as the minimal requirement of a set of sufficient mutation operators for mutation testing. Note that Wong and Mathur's 2 mutation operators are among Offutt et al.'s 5 mutation operators. Barbosa et al. [4] proposed six guidelines to determine sufficient mutation operators. The application of some of the six guidelines requires compilation and execution of a large number of mutants. Based on the six guidelines, Barbosa et al. determined 10 mutation operators, and found that, using the same way as Offutt et al. to measure the effectiveness, the effectiveness values of the 10 mutation operators on 27 subjects (which were also used to determine the 10 mutation operators) are between 95.8% and 100%, with the average of 99.6%. Siami Namin et al. [38] leveraged variable reduction to determine 28 mutation operators using the execution information of a subset of mutants. As their work aims at mutation analysis rather than mutation testing, they evaluated the 28 mutation operators on the Siemens programs only in the context of mutation analysis. They did not provide evidence about whether the 28 operators are sufficient in mutation testing.

Compared with studies on operator-based mutation selection, studies on random mutant selection, which Acree et al. [1] first proposed in 1979, are limited. Wong and Mathur [41, 42] empirically studied the technique of randomly selecting 10% to 40% mutants generated with 22 mutation operators in Mothra. Barbosa et al. [4] used random mutant selection as a control technique when evaluating their 10 mutation operators. In their study, Barbosa et al.'s 10 mutation operators are more effective than random mutant selection. Our study differs from previous studies on random mutant selection for mutation testing as follows. First, our study investigates some operator-based techniques (i.e.,Offutt et al.'s technique [32] and Siami Namin et al.'s technique [38]) previously not empirically compared with random mutant selection, and our results on Barbosa et al.'s technique [4] contradict previous results. Second, our study investigates two random mutant-selection techniques, while previous studies investigated only one random mutant-selection technique. Third, our study uses larger subjects than previous studies on random mutant selection. Finally, our study investigates both average effectiveness and standard deviation of effectiveness, while previous studies investigate only average effectiveness.

6. CONCLUSION AND FUTURE WORK

In this paper, we report an empirical study attempting to answer one important open question in the field of mutant selection for mutation testing. Our experimental results show that none of the three experimented operator-based mutant-selection techniques is superior to random mutant selection in terms of either effectiveness or stability. Furthermore, random mutant selection can still achieve competitive results when selecting much fewer mutants than each operator-based mutant-selection technique.

In future work, we plan to investigate three main issues. First, as Siami Namin et al.'s mutation operators (which are determined for mutation analysis) are quite effective in mutation testing, we are thus interested in whether random and operator-based mutant-selection techniques for mutation testing are also effective in the context of mutation analysis. Second, we plan to extend our experiments to other and larger subjects to further corroborate the findings in our study. Finally, we also plan to investigate new techniques of mutant selection on the basis of individual mutants, which our study in this paper has shown to be a promising direction.

7. REFERENCES

[1] A. T. Acree, T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Mutation analysis. Technical report, Georgia Institute of Technology, 1979.

[2] H. Agrawal, R. A. DeMillo, B. Hathaway, W. Hsu, W. Hsu, E. W. Krauser, R. J. Martin, A. P. Mathur, and E. Spafford. Design of mutant operators for the C programming language. Technical report, Purdue University, 2006.

[3] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *Proc. ICSE*, pages 402–411, 2005.

[4] E. F. Barbosa, J. C. Maldonado, and A. M. R. Vincenzi. Toward the determination of sufficient mutant operators for C. *Software Testing, Verification and Reliability*, 11(2):113–136, 2001.

[5] L. C. Briand, Y. Labiche, and M. M. Sówka. Automated, contract-based user testing of commercial-off-the-shelf components. In *Proc. ICSE*, pages 92–101, 2006.

[6] L. C. Briand, Y. Labiche, and Y. Wang. Using simulation to empirically investigate test coverage criteria based on statechart. In *Proc. ICSE*, pages 86–95, 2004.

[7] T. A. Budd, R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Theoretical and empirical studies on using program mutation to test the functional correctness of programs. In *Proc. POPL*, pages 220–233, 1980.

[8] M. E. Delamaro, J. C. Maidonado, and A. P. Mathur. Interface mutation: An approach for integration testing. *IEEE TSE*, 27(3):228–247, 2001.

[9] M. E. Delamaro and J. C. Maldonado. Proteum - a tool for the assessment of test adequacy for C programs. In *Proc. Conference on Performability in Computing Systems*, pages 79–95, 1996.

[10] R. A. DeMillo, E. W. Krauser, and A. P. Mathur. Compiler-integrated program mutation. In *Proc. COMPSAC*, pages 351–356, 1991.

[11] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, 1978.

[12] R. A. DeMillo and R. J. Martin. The mothra software testing environment user's manual. Technical report, Software Engineering Research Center, 1987.

[13] R. A. DeMillo and A. J. Offutt. Constraint-based automatic test data generation. *IEEE TSE*, 17(9):900–910, 1991.

[14] H. Do and G. Rothermel. On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE TSE*, 32(9):733–752, 2006.

[15] S. Elbaum, A. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. In *Proc. ISSTA*, pages 102–112, 2000.

[16] J. B. Goodenough and S. L. Gerhart. Toward a theory of test data selection. *IEEE TSE*, 1(2):156–173, 1975.

[17] B. Grün, D. Schuler, and A. Zeller. The impact of equivalent mutants. In *Proc. International Workshop on Mutation Analysis*.

[18] R. G. Hamlet. Testing programs with the aid of a compiler. *IEEE TSE*, 3(4):279–290, 1977.

[19] R. Hierons and M. Harman. Using program slicing to assist in the detection of equivalent mutants. *Software Testing, Verification and Reliability*, 9(4):233–262, 1999.

[20] W. E. Howden. Weak mutation testing and completeness of test sets. *IEEE TSE*, 8(4):371–379, 1982.

[21] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc. ICSE*, pages 191–200, 1994.

[22] J. A. Jones and M. J. Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proc. ASE*, pages 273–282, 2005.

[23] E. W. Krauser, A. P. Mathur, and V. Rego. High performance testing on SIMD machines. In *Proc. Second Workshop on Software Testing, Verification, and Analysis*, pages 171–177, 1988.

[24] Z. Li, M. Harman, and R. Hierons. Search algorithms for regression test case prioritization. *IEEE TSE*, 33(4):225–237, 2007.

[25] M.-H. Liu, Y.-F. Gao, J.-H. Shan, J.-H. Liu, L. Zhang, and J.-S. Sun. An approach to test data generation for killing multiple mutants. In *Proc. ICSM*, pages 113–122, 2006.

[26] Y.-S. Ma, J. Offutt, and Y. R. Kwon. MuJava : An automated class mutation system. *Software Testing, Verification and Reliability*, 15(2):97–133, 2005.

[27] A. P. Mathur. Performance, effectiveness, and reliability issues in software testing. In *Proc. COMPSAC*, pages 604–605, 1991.

[28] J. Mayer and C. Schneckenburger. An empirical analysis and comparison of random testing techniques. In *Proc. International Symposium on Empirical Software Engineering*, pages 105–114, 2006.

[29] E. Mresa and L. Bottaci. Efficiency of mutation operators and selective mutation strategies: An empirical study. *Software Testing, Verification and Reliability*, 9(4):205–232, 1999.

[30] A. J. Offutt and W. M. Craft. Using compiler optimization techniques to detect equivalent mutants. *Software Testing, Verification and Reliability*, 4(3):131–154, 1994.

[31] A. J. Offutt, Z. Jin, and J. Pan. The dynamic domain reduction approach to test data generation. *Software Practice and Experience*, 29(2):167–193, 1999.

[32] A. J. Offutt, A. Lee, G. Rothermel, R. H. Untch, and C. Zapf. An experimental determination of sufficient mutation operators. *ACM TOSEM*, 5(2):99–118, 1996.

[33] A. J. Offutt and J. Pan. Automatically detecting equivalent mutants and infeasible paths. *Software Testing, Verification and Reliability*, 7(3):165–192, 1997.

[34] A. J. Offutt, R. P. Pargas, S. V. Fichter, and P. K. Khambekar. Mutation testing of software using mimd computer. In *Proc. International Conference on Parallel Processing*, pages 257–266, August 1992.

[35] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Proc. ICSM*, pages 34–43, 1998.

[36] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Test case prioritization: an empirical study. In *Proc. ICSM*, pages 179–188, 1999.

[37] D. Schuler, V. Dallmeier, and A. Zeller. Efficient mutation testing by checking invariant violations. In *Proc. ISSTA*, pages 69–80, 2009.

[38] A. Siami Namin, J. H. Andrews, and D. J. Murdoch. Sufficient mutation operators for measuring test effectiveness. In *Proc. ICSE*, pages 351–360, 2008.

[39] J. Tuya, M. J. Suárez-Cabal, and C. de la Riva. Mutating database queries. *Information and Software Technology*, 49(4):398–417, 2007.

[40] R. Untch, A. J. Offutt, and M. J. Harrold. Mutation analysis using program schemata. In *Proc. ISSTA*, pages 139–148, 1993.

[41] W. E. Wong. *On Mutation and Data Flow*. PhD thesis, Purdue University, December 1993.

[42] W. E. Wong and A. P. Mathur. Reducing the cost of mutation testing: An empirical study. *The Journal of Systems and Software*, 31(3):185–196, 1995.

[43] M. R. Woodward and K. Halewood. From weak to strong, dead or alive? An analysis of some mutation testing issues. In *Proc. Second Workshop on Software Testing, Verification, and Analysis*, pages 152–158, 1988.

[44] H. Zhu. A formal analysis of the subsume relation between software test adequacy criteria. *IEEE TSE*, 22(4):248–255, 1996.