# A Test Suite Reduction Method based on Test Requirement Partition

Wan Yongbing[1], Xu Zhongwei[1], Yu Gang[2] and Zhu YuJun[1]

[1]*School of Electronics & Information Engineering, Tongji University, Shanghai, China*
[2]*Sydney Institute of Language and Commerce, Shanghai University Shanghai, China*

*(wybingsh, xuzhongweish, yugang509,dyzhuyujun)@163.com*

***Abstract***

*Test suite reduction aims at improving the effectiveness of testing and cutting down the test cost with the least test cases under the condition of satisfying all testing objectives. This paper proposes a new method for test suite reduction with test requirement partition. First, it gives a partition to the set of all the available test cases based on the test requirements. After that, a test suite by the partition is generated and then a smaller test suite is obtained by further reduction according to the relationship between test requirements and test suites. At last, the experimental result shows that the proposed approach is helpful to generate the reduced test suite, which is used to test all the test requirements sufficiently by comparing with the existed methods, at the cost of a moderate loss in fault detection capability.*

***Keywords:*** *Test suite reduction, Test requirement, Test case, Fault detection capability*

## 1. Introduction

Testing is one of the most important techniques to validate the quality of software systems, and if it is used in an effective way, it can provide important evidences of reliability of a product. Software testing is an expensive activity. However it consumes a large part of effort and cost. Practically during regression testing, the size of the test suite has a large impact on the total costs in order to facilitate the testing of evolving software. So the question of test suite reduction is to find a minimal subset of the test suite that is still fulfills the given test requirements.

There are a lot of different techniques to approximate a minimal subset of the test suite. Those techniques try to minimize the size of the regression test suite by identifying redundant test suite based on some coverage criteria or by making use of traceability matrix. Even though these approaches can certainly reduce the number of test cases in a test suite, they have two major drawbacks: on the one hand, they are incapable of handing large size test suites because of computational complexity of test case comparisons; on the other hand, the overall ability to detect fault is reduced while minimizing a test suite, which are still satisfying the test requirements. Previous studies have shown that sometimes the number of test suite reduction is small, but sometimes this reduction is significant. So there has to be a novel approach which should be capable of doing it in a moderate manner.

In this paper, we propose a new approach to test suite reduction. The key step of our approach is that obtaining a partition to the set of all the available test cases based on the test requirements, we then generate a smaller test suite by further reduction according to the relationship between test requirements and test suites. We implemented our approach and conducted experiments with several programs to evaluate and compare the effectiveness of

our approach with prior experimental studies on the test suite reduction. The main contributions of this paper are as follows: Firstly, a novel yet simple approach to test suite reduction that focuses on retaining test cases that may expose different faults in software testing. Secondly, the experimental results clearly show the potential of our new reduction approach is helpful to generate the reduction test suite to test all the test requirements sufficiently by comparing with the existed methods.

The remaining paper is organized as follows: Section 2 first introduces the problem of test suite reduction, and then discusses related work. Section 3 presents a new approach of test suite reduction, and relevant algorithm for test suite reduction is described. In Section 4, we present an experimental study to evaluate the effectiveness of the test suite reduction algorithm. Finally, the conclusions are mentioned in Section 7.

## 2. Related Work

When software is re-tested after some modifications, the costs of running a complete test suite against the software repeatedly can be quite high. Generally speaking, not all test cases of a test suite are necessary to achieve the given test requirements. Therefore, the aim of test suite reduction is to remove test cases from a test suite in such a way that reduced test suite should satisfy the three conditions, including provide the same coverage of the software as the original test suite, fulfill the same requirements as the original test suite, as well as be capable to reveal the same set of faults as the original test suite.

### 2.1. Research Question

The first formal definition of the test suite reduction problem is introduced in 1993 by Harrold et al. as follows:

**Given:** $\{t_1, t_2, \ldots, t_m\}$ is a test suite $\boldsymbol{T}$ from $m$ test cases and $\{r_1, r_2, \ldots, r_n\}$ is a set of test requirements that must be satisfied in order to provide desirable coverage of the program and each subset $\{T_1, T_2, \ldots, T_n\}$ from $\boldsymbol{T}$ are associated with each of the $r_i s$ such that each test case $t_j$ belonging to $T_i$ can be used to test $r_i$.

**Problem:** Find a minimal test suite $T_i$ from $\boldsymbol{T}$ which satisfies all $r_i s$ covered by original suite $\boldsymbol{T}$.

The requirements $r_i$ can represent any test case requirements, and a representative set of test case must contain at least one test case from each subset $T_i$. Theoretically speaking, the problem of finding the minimal subset $T'$, $T' \in T$ which satisfies all requirements of $\boldsymbol{T}$, is NP-complete. Thus, several heuristics approaches to solve this problem have been presented.

Intuitively, test suite reduction will generate a new test suite, which only the relevant subset remains and the others are discarded [1-3]. On the other hand, the issues of fault detection efficiency, which strongly related to test suite reduction problem caused more attention for the researchers [4, 5].

### 2.2. Literature Review

A Classical greedy heuristic (G) [6] solves the problem by recursive action as follows: Pick the test cases from a test suite and find out whether its satisfy the most requirements; remove all the requirements covered by the selected test cases, and repeat the process until all the

requirements are satisfied. The ties are broken arbitrarily. Several algorithm based on G algorithm have been proposed for test case selection and prioritization from a large test suite. Chen and Lau in [8, 9] described two techniques for dividing a test suite into *k* smaller sub-suites such that if optimally reduced suite. However, these two dividing approaches can not be applied to every suite. Another heuristic (HGS) to optimize test suites developed by Harrold and Gupta [7]: Select a representative set of test cases from a test suite until satisfied a requirement by a test case. This approach will take additional computation for selection the next test case.

Harder and Mellon in [10] proposed a minimization approach that uses an operational abstraction, which is a formal specification for software derived from actual behavior. This idea is to keep the tests that change the operational abstraction and remove those tests that do not change the operational abstraction. The approach described two strategies in [11] for test suite reduction that are tailored to be used specifically with the modified condition/decision coverage (MC/DC) criterion. Black et al in [12] presented a "bi-criteria" approach for test suite reduction that considers not only the coverage information for the test, but also whether or not each test exposes a particular fault. This approach aims to compute optimally reduced suites containing the most fault-revealing tests.

In [13] the authors propose testing requirement reduction method to generate the reduced testing requirement set, which is the basis of test suite generation, reduction and optimization. A reduced test suite generation approach based on predicate abstraction was proposed in [14], which divided the status space of the software model according to the given predicates to get the equivalence classes. The reduced abstract status was settled using the mapping between the status sets, and generated the reduced test suite based on the transition of the status. Jeffrey and Gupta in [15] presented a approach for test suite reduction that attempts to use additional coverage information of test cases to selectively keep some additional test cases in the reduced suites that are redundant with respect to the testing criteria used for suite reduction, with the goal of improving the fault detection capability of the reduced suites.

However, the above approaches we can not exactly said which one gives better test suites reduction by they lack of fault detection capability. In this paper we present a novel approach to test suite reduction which does not have a negative influence on the ability of fault detection. Some of the basic definitions and notation are introduced in the following.

## 3. Test Suite Reduction based on Test Requirement Partition

Where there is great love, there are always miracles. Love is like a butterfly. It goes where it pleases and it pleases where it goes. If I had a single flower for every time I think about you, I could walk forever in my garden. Within you I lose myself, without you I find myself wanting to be lost again. At the touch of love everyone becomes a poet.

### 3.1. Concept and Notation

For any set *A*, we use $|A|$ to represent the cardinality of *A*. We use *R* and *T* to denote the set of all requirements and its test suite, respectively. Test cases in *T* satisfying requirements in *R* can be described as a binary relation $S(T,R)$ from *T* to *R*, known as the satisfiability relation [16].

**Definition 1**. Let *R* be the set of all requirements and *T* be a test suite. For any test case $t \in T$ and any requirement $r \in R$, the satisfiability relation $S(T,R)$ from *T* to *R* is defined as $S(T,R) = \{(t, r) \in T \times R \mid t \text{ satisfies } r\}$.

For each $t \in T$ and $T_1 \in T$, the set of all requirements satisfied by $t$ and $T_1$ are described as $Req(t)=\{ r \in R \mid (t,r) \in S(T,R)\}$ and the set of all test cases in $T$ satisfying $r$ is described as $Test(r)=\{ t \in T \mid (t,r) \in S(T,R)\}$. There are some properties of *Req* under the followings:

**Lemma 1**. Let $S$ be the safisfiability relation between $T$ and $R$, and $T_1$, $T_2 \in T$ :

(1) if $T_1 \subseteq T_2$, then $Req(T_1) \subseteq Req(T_2)$;

(2) $Req(T_1 \bigcup T_2) = Req(T_1) \bigcup Req(T_2)$;

(3) $Req(T_1 \bigcap T_2) = Req(T_1) \bigcap Req(T_2)$;

(4) $Req(T_1) \backslash Req(T_2) \subseteq Req(T_1 \backslash T_2)$.

**Definition 2**. Let $T_1$, $T_2 \in T$, if and only if $T_1 \subseteq T_2$ and $T_2 \subseteq T_1$, namely two set have the same elements, they are defined as the equality of sets; if and only if $T_1 \bigcap T_2 = T_1$, namely everything in $T_1$ is also in $T_2$, they are defined as $T_1$ is a subset of $T_2$; if and only if $T_1 \bigcap T_2 \neq \varnothing$, namely, they are defined as the union of sets; if and only if $T_1 \bigcap T_2 = \varnothing$, they are defined as the intersection sets.

**Definition 3**. Let $S$ be the safisfiability relation between $T$ and $R$, and $T_1 \in T$, $T_1$ is said to be a representative set of $S$ if $Req(T_1)=R$, as well as the set $T_1$ of $S$ is a optimal set if $| T_1| \leq | T_2|$ for every representative set $T_2$ of $S$.

We will here use $REP(S)$ to represent the sets of all representative set and $OPT(S)$ to represent the sets of all optimal presentative sets of $S$ respectively. Obviously, $OPT(S) \subseteq REP(S)$ and both are non-empty.

**Definition 4**. Let $S$ be the satisfiability relation between $T$ and $R$, and $t \in T$, $t$ is said to be redundant with respect to $S$ if $R=Req(T\backslash\{t\})$.

If $t$ is redundant, $T\backslash\{t\}$ still satisfies $R$. A natural approach is to divide $S(T,R)$ into $S(\{t\},\varnothing)$ and $S(T\backslash\{t\},R)$. However, such a reduction strategy does not guarantee the reconstruction of optimal representative sets of $S(T,R)$ as illustrated in Example 1.

**Example 1**. The satisfiability relation $S$ between $\{t_1, t_2, \ldots, t_6\}$ and $\{r_1, r_2, \ldots, r_7\}$ is given in Table 1. The symbol "1" denotes a test case satisfies a requirement, while "0" denotes not.

**Table 1. The satisfiability relation *S***

| | Cardinality | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $r_6$ | $r_7$ |
|---|---|---|---|---|---|---|---|---|
| $t_1$ | 2 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| $t_2$ | 3 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| $t_3$ | 2 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| $t_4$ | 3 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| $t_5$ | 3 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| $t_6$ | 3 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |

We have $t_4$ being redundant and $OPT(S)=\{\{t_1, t_4, t_5\}\}$. After dividing $S(T,R)$ into $S(T\backslash\{t_4\}, R)$ and $S(\{t_4\}, \varnothing)$, we have: $OPT(S(T\backslash\{t_4\},R))=\{\{t_1, t_2, t_3, t_5\},\{ t_1, t_2, t_3, t_6 \}\}$, $OPT(S(\{t_4\},\varnothing))=\{\varnothing\}$. Obviously, there is no way to construct $\{t_1, t_4, t_5\}$ from $\{\{t_1, t_2, t_3, t_5\}, \{t_1, t_2, t_3, t_6\}\}$. Therefore, a test suite reduction model to deal with this issue is introduced in the next section.

### 3.2. Concept and Notation

Our approach to test suite reduction uses the set theory from that we find the intersection between the one requirement to another requirement of branch coverage criteria for the set of test cases. If only take the original test suites as the reduction objects, not take the relationship between test requirements into account, it may clearly generate an optimal set for reduced test suites. Suppose $T = \bigcup T_i$ ( $i$=1,2,…,$n$) are the original test suites, that is $T$ are covered by $T_i$ ( $i$=1,2,…,$n$), therefore the reduced objects are concentrated on the overlapped portion of $T_i$. Here are some reduction principles as following:

(1) **Subset:** If $T_i \subseteq T_j$, then $T_i = Test(r_i)$, $T_j = Test(r_j)$), the test suites selected from $T_i$ is always at the same time to meet the test requirements $r_i$ and $r_j$. So, $T_i$ may reduce from the sets.

(2) **Set intersection:** If $T_i$ and $T_j$ have the same elements $t$, according to definition 2, as for the test suite $t \in T_i \bigcap T_j$, also satisfying the test requirements $r_i \bigcap r_j$, that means only the test suite $t \in Test(r_i \bigcap r_j)$ need to reserve when test suites reduction. So, it may reduce as $T_i \bigcap T_j \setminus \{t \mid t \in T_i \ (or\, T_j) \text{ and } t \in Test(r_i \bigcap r_j)\}$.

(3) **Set union:** If $T_i$ and $T_j$ are incompatible, that is to say there is not an element $t$ belongs to both $T_i$ and $T_j$. So $T_i$ and $T_j$ are all need to reserve in this circumstance.

(4) **Set equality:** If all of the elements $t$ in $T_i$ and $T_j$ are same, then only one between $T_i$ and $T_j$ have to reserve.

### 3.3. Test Suite Reduction Algorithm

Suppose the test purpose of a certain program is made up of $n$ test requirements $r_1, r_2,\ldots, r_n$. Accordingly, the possible test suite corresponding to each test requirement $r_i$ ($1 \le i \le n$) is $T_i$ ($T_i \neq \varnothing$). That is to say any test cases in $T_i$ can be full testing coverage for testing requirement $r_i$. If $T_i = \varnothing$ ( $i$=1,2,…,$n$), then the testing requirement $r_i$ is called infeasible. In this paper, we assume that all the test requirements are feasible and non-empty. Here let $T = T_1 \bigcup T_2 ... \bigcup T_n$ represents the entire possible test suites.

**Algorithm 1:** Partition algorithm for all of the possible test suites $T$

**Input:** original possible test suite $T$
**Output:** a partition suite of test cases from the original test suite: $RS = \{T_1, T_2, \ldots, T_n\}$

begin:
1: *index* = 1; *num* = | $T$ |; $RS = T_1$; $RS_1 = RS_2 = Null$; /* initialization */
2: while (*index* < *num*) do
3: $RS_1 = RS$ ;
4: while ($RS_1 \neq Null$) do

```
5:    if (RS1.test ∩ Tindex ≠ ∅ ) then
6:       RS2.test = RS1.test ∩ Tindex ;
7:       RS2.mark = RS1.mark ∪ {index} ;
8:       Tindex = Tindex \ RS2.test ;
9:       RS1.test = RS1.test \ RS2.test ;
10:      RS2.next = RS1.next ;
11:      RS1.next = RS2 ;
12    end-if
13:   if (Tindex ≠ ∅ )then
14:      RS2.test = Tindex ;
15:      RS2.mark = {index} ;
16:      RS2.next = null ;
17:      RS.next = RS2 ;
18:    end-if
19:  end-while
20: end-while
21: return RS
end
```

The common method is selecting the relevant test cases to form a test suite, and then using the existing approach for further reduction. The disadvantage of those methods is not considering the relationship between test requirements comprehensively when selecting the relevant test cases. Consequently, the test suite can not be optimal achieved fundamentally. Taking it into account, the relationship between test requirements should be clearly sort out, before selecting test cases, in each test purpose. In fact, that is a partition for all of the possible test suites *T*. In this paper, we present a partition algorithm for all of the possible test suites *T*, as shown in Algorithm 1.

In algorithm 1, we use a list to save the result in order to obtain a partition of all possible test suites *T*. Each subset exists in a node, and only ensures that each test case stored in a node is disjoint can obtain the total partition of *T*. The running time of Algorithm 1 using the asymptotic notation is $O(n*|CT|)$, where $|CT|$ is the cardinality of the original test suite, and $n$ is the number of test requirement.

In the output of algorithm 1, we can see which test requirement can be satisfied by the same test cases, and each test case can satisfy which test requirements at most. After that, a minimal test suite can be obtained by using the combination method of Greedy algorithm and linear search, according to the results of algorithm 1. Based on the previously given model in Section 3.2, we propose a test suite reduction algorithm, as shown in Algorithm 2.

**Algorithm 2:** Test suite reduction algorithm

**Input:** a partition suite of test cases $T_i$ (generated by the process of algorithm 1)
**Output:** a reduced suite of test cases *RT*

```
begin:
1: T" = T ; T' = ∅ ; RT = ∅ ; /* initialization */
2: while ( T" = ∅ )
3:  Tk = max{{| Ttemp | | Ttemp ⊆ T" }} ; /* selected the largest cardinality of test suites
*/
4:  T" = T" \ Tk ;
5:  T' = Tk
```

6: for each $T_i \in T''$ do
7: if ($T' \subseteq T_i$) then /* deal with the Subset */
8: $T'' = T'' \setminus T'$;
9: else if ($T' \supseteq T_i$) then /* deal with the Set equality */
10: $T'' = T'' \setminus T_i$;
11: end-if
12: end-if
13: if ($T' \cap T_i$) then; /* deal with the Set intersection */
14: $T' = \{t \mid t \subseteq T' \text{ and } t \subseteq Test(r_k \cap r_j)\}$;
15: $T'' = T'' \setminus T'$;
16: end-if
17: end-for
18: $RT = RT \cup TC''$
19: end-while
end

In algorithm 2, we obtain an optimized test suite by using linear search method to reduce the test suites scale (line 6). Then remove the reduced test suites from the original test suites, according to the reduction principle of inclusiveness and equivalence relations (line 7). After that, we continue reducing test suite with intersecting relation, which is to delete the test cases belonging to the two test suite, but not all. Only delete the intersection part of the test cases and save those who can satisfy the intersection of two test requirements (line 13). At the end output the test suites after reduction (line 18).

Algorithm 2 is based on two loops in order to process all steps of all partition suites of test cases $T_i$. Thus the running time of Algorithm 2 is $O(n^2)$, where $n$ represents the number of test requirement which a test case can satisfy it at most.

## 4. Experimental Study

In order to evaluate the effectiveness of the proposed algorithms for test suite reduction, a number of experimental studies are conducted in this section. To compare the empirical results with prior studies, we implemented prototypes of the reduction algorithms. Both are written in turbo C programs. We performed the studies on a 2.5GHz, PentiumG645T, 2GB RAM computer and running Windows XP Professional. Our Reduction algorithm implemented in C++ and the process of test suite reduction was implemented as a set of scripts in the interpreted Ruby 2.0 language.

Our subject for the study is the *ping* programs, a tool for network supervision, which consists of 1479 lines of C code. We also used 6 faulty versions of the program, each containing several faults: (1) changing the operator in an expression; (2) changing the value of a variable; (3) adding some code; (4) removing some code; (5) changing the logical behavior of the code. However, the base version contains no faults, which is assumed for the aim of studies. We also have a test pool of 500 test cases, this subject and these test suites have been used in similar studies [14].

As for the question of test suite reduction is NP-complete, most of the studies to acquire the approximate values by heuristic. Therefore, to evaluate our test suite reduction algorithm, we applied our test suite reduction prototypes to each of the 500 test suites, by comparing the results with *G*、*HGS* and *GRE*, respectively. The results of this experiment are shown in Table 2.

**Table 2. The results of the experiments comparing**

| program | Lines of Code | RTC-G | RTC-HGS | RTC-GRE | RTC-Our Approach | %Size Reduction | %Fault Loss |
|---|---|---|---|---|---|---|---|
| Ping-2.17-0 | 1479 | 427 | 413 | 409 | 366 | 26.8 | 10.32 |
| Ping-2.17-1 | 1479 | 429 | 414 | 412 | 370 | 26.0 | 9.91 |
| Ping-2.17-2 | 1479 | 433 | 413 | 410 | 367 | 26.6 | 11.08 |
| Ping-2.17-3 | 1508 | 435 | 421 | 418 | 376 | 24.8 | 8.45 |
| Ping-2.17-4 | 1380 | 422 | 415 | 393 | 357 | 28.6 | 13.23 |
| Ping-2.17-5 | 1479 | 430 | 416 | 411 | 368 | 26.4 | 10.96 |

As the results shown, *G*, *HGS* and *GRE* have the same performance roughly in the generation of test cases reduction, also in line with the comparison of the results in [17]. In Table 1, the values of *RTC* indicates that our approach may generate less test suites than existing techniques under the same test requirements, and *%* also represents that our approach performed better.

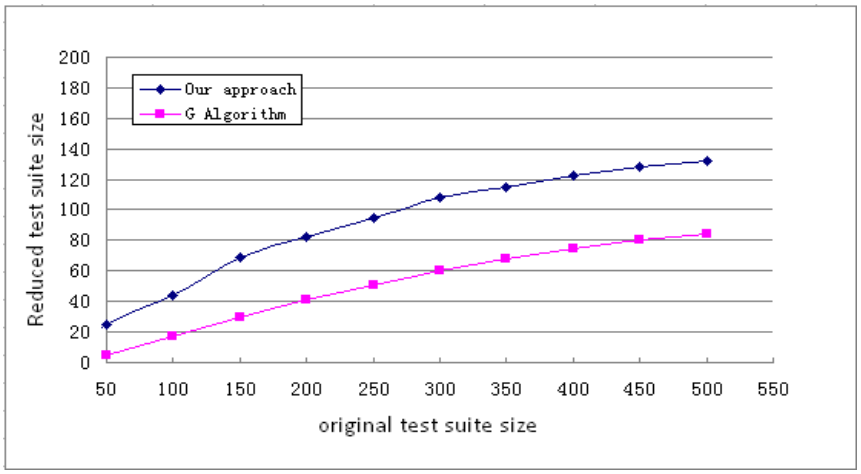



**Figure 1. Size of the reduced test suite**

In the cost of algorithm execution, the time complexity of our approach is $O(k^2)$ in the step of initial partition of test suites and the step of test suites reduction. The total algorithm complexity is $O(k^2)$, where $k$ represents the number of test requirement a test case can satisfy at most. However, the existing approaches of *G*, *HGS* and *GRE* are $O(mn*\min(n,m))$, $O(m(m+n)*k)$ and $O(\min(m,n)(m+(n^2*k)))$, respectively. Figure 2 shows that more test cases increase more effectiveness in time consuming.


**Figure 2. Time to run the reduction algorithm**

Because some studies have indicated that there is fault detection ability loss in a reduced test suites, here we also considered the fault detection ability lost caused by each test suite when the test suite was reduced using our approach. In order to analysis the fault detection ability, we compared the ability of the reduced test suite *T* with the different faulty versions of program *Ping*. As Figure 3 shows, comparing with the original suites, the average percentage reduction in fault detection for all of the reduced test suites is approximate 10.6%.

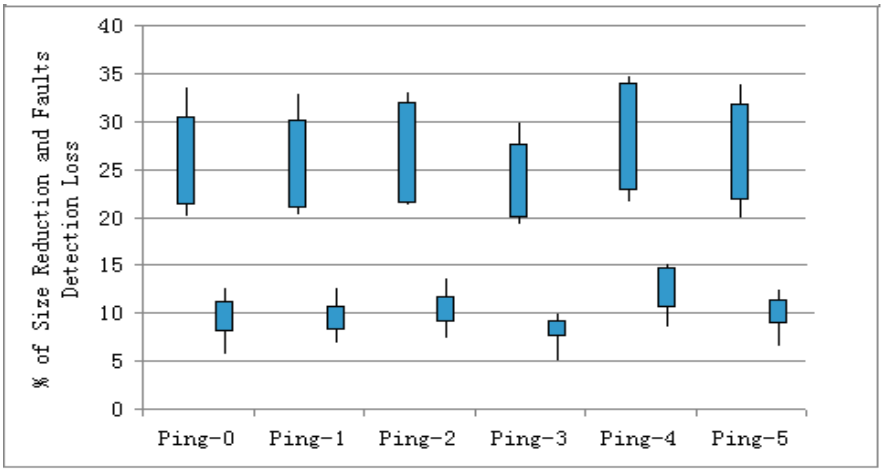



**Figure 3. Percentage reduction in fault detection**

## 5. Conclusion

In this paper, we introduce an approach to reduce the size of a test suite based on test requirement dividing, with the goal of decreasing the loss of fault detection effectiveness. This approach is general and can be integrated into some existing test suite reduction algorithm. In order to understand the performance of the new approach comprehensively, *G*,*HGS* and *GRE*, an experimental study is being conducted with *Ping* program. The results

indicate that the reduction is significant. And in contrast to previous approaches, our approach always performed better and less fault detection ability loss at the same time.

Although our initial studies are encouraging, a number of different programs, preferably of different sizes, even from different domains, must be conducted by using our test suite reduction algorithm. In future work, we plan to evaluate the feasibility of our approach for some practical application.

## Acknowledgements

The authors are grateful to Professor Yu Gang of the Shanghai University for many interesting discussions about the topic of our article. The authors also thank the anonymous referees for their thoughtful reviews that helped in meaningfully improving the initial version of this paper. Finally, this research is partially supported by the National Natural Science Foundation of China (Grant No.61075002).

## References

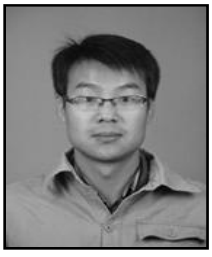
[1] G. Rothermel, R. H. Untch and M. J. Harrold, "Prioritizing Test Cases for Regression Testing", IEEE Trans. Software Engineering, vol. 10, **(2001)**, pp. 27.
[2] G. Rothermel, M. J. Harrold and C. Hong, "Empirical Studies of Test Suite Reduction", Software Testing, Verification and Reliability, vol. 4, **(2022)**, pp. 12.
[3] Z. Y. Chen, B. W. Xu, X. F. Zhang and C. H. Nie, "A Novel Appproach for Test Suite Reduction Based on Requirement Relation Contraction", Proc. of the 23rd ACM Symposium on Applied Computing, ACM, **(2008)** March, 1-5; Fortaleza, Brazil.
[4] W. E. Wong, J. R. Horgan and S. London, "Effect of Test Set Minimization on Fault Detection Effectiveness", Software, Practice and Experience, vol. 4, **(1998)**, pp. 28.
[5] S. Tallam and N. Gupta, "A Concept Analysis Inspired Greedy Algorithm for Test Suite Minimization", Proc. of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. ACM, **(2005)** October 1-8, Lisbon, Portugal.
[6] V. Chivtal, "A Greedy Heuristic for the Set Covering Problem", Mathematics of Operation Research, vol. 3, no. 4, **(1979)**.
[7] M. J. Harrold, R. Gupta and M. L. Sofa, "A Methodology for controlling the size of a test suite", ACM Transcations on Software Engineering and Methodology, vol. 3, no. 2, **(1993)**.
[8] T. Y. Chen and M. F. Lau, "A new heuristic for test suite reduction", Information and Software Technology, vol. 5, **(1998)**, pp. 40.
[9] T. Y. Chen and M. F. Lau, "On the divide and conquer approach towards test suite reduction", Information and Software Technology, vol. 1, **(2003)**, pp. 152.
[10] M. Harder, J. Mellen and M. D. Ernst, "Improving Test Suites via Operational Abstraction", Proc. of the 25th International Conference on Software Engineering, **(2003)** June, 60-71, Washington, USA.
[11] J. A. Jones and M. J. Harrold, "Test Suite Reduction and Prioritization for Modified Condition/ Decision Coverage", IEEE Trans. on Software Engineering, vol. 3, **(2003)**, pp. 29.
[12] J. Black, E. Melachrinoudis and D. Kaeli, "Bi-Criteria Models for All-Uses Test Suite Reduction", Proc. of the 26th International Conference on Software Engineering, **(2004)** May, pp. 106-115, Boston, USA.
[13] X. F. Zhang, B. W. Xu and C. H. Nie, "An Approach for Optimizing Test Suite Based on Testing Requirement Reduction", Journal of Software, vol. 4, **(2007)**, pp. 18.
[14] X. Guo and H. G. Zhang, "Approach for reduced test suite generation based on predicate abstraction", Journal on Communications, vol. 3, **(2012)**, pp. 33.
[15] D. Jeffrey and N. Gupta, "Improving Fault Detection Capability by Selectively Retaining Test Cases during Test Suite Reduction", IEEE Trans. on Software Engineering, vol. 2, **(2007)**, pp. 33.
[16] T. Y. Chen and M. F. Lau, "Dividing Strategies for the Optimization of a Test Suite", Information Processing Letters, vol. 3, **(1996)**, pp. 60.
[17] T. Y. Chen and M. F. Lau, "A Simulation Study on Some Heuristics for Test Suite Reduction", Information and software Technology, vol. 13, **(1998)**, pp. 40.

# Authors

**Yongbing Wan.** He received his BEng in Information Science and Technology (2003) from Nanchang University and M.S in Computer Sciences (2009) from Jiangxi Normal University. Since 2009 he is a PhD candidate of computer Science at computer science and technology Department, Tongji University. His current research interests include different aspects of Formal Modeling, Simulation and Testing for Safety-critical Systems.

**Zhongwei Xu.** He received his M.Sc. in Computer Science (1990) and PhD in Information Sciences (2000) from Tongji University. Now he is full professor in school of electronic and information engineering of Tongji University. His current research interests include different aspects of Artificial Intelligence and Safety-critical Systems.

**Gang Yu.** He received BEng degree in 1999 and a Master's degree in 2002 from Jiangxi Normal University, P R China. He is a lecturer in Sydney institute of language and commerce of Shanghai University. He is now a PhD graduate student in Tongji University. His research interests include different aspects of Formal Modeling, Simulation and Testing for Safety-critical Systems.



**Yujun Zhu.** He received his BEng in Computer Science (2006) from Qidao Univeristy and M.Sc in Computer Sciences (2002) from Anhui Engineering University. Since 2009 he is a PhD candidate of Computer Science at computer science and technology Department, Tongji University. His current research interests include different aspects of Simulation and Testing for Safety-critical Systems.