A Novel Approach for Test Suite Reduction Based on Requirement Relation Contraction

Zhenyu Chen, Baowen Xu, Xiaofang Zhang, Changhai Nie
School of Computer Science and Engineering
Southeast University, Nanjing, China
{zchen, bwxu, xfzhang, changhainie}@seu.edu.cn

ABSTRACT

The goal of test suite reduction is to satisfy all testing requirements with the minimum number of test cases. Existing techniques can be applied well on the constructed test suite. However, it is possible and necessary to optimize testing requirements before test case generation. In this paper test suite reduction is solved by testing requirement optimization. A requirement relation graph is proposed to minimize the requirement set by graph contraction. An experiment on specification-based testing is designed and implemented. The empirical studies show that the testing requirements can be optimized by the graph contraction methods effectively.

Keywords

test case reduction, testing requirement optimization, graph contraction

1. INTRODUCTION

Software testing is an important and expensive activity in the whole lifecycle of software development. In software testing, testing objectives are defined first and considered as a set of testing requirements. The testing requirements are always extracted from software specifications or implementations. Once a set of requirements is determined, a set of test cases, called test suite, is designed and generated to satisfy the requirements manually or automatically [6, 7, 13].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2008 ACM 978-1-59593-753-7/08/0003 ...$5.00.

In practice, test cases designed for a particular requirement may also satisfy other requirements, and a requirement may be satisfied by more than one test case. As a result, the constructed test suite may contain redundancy, and some subsets of the constructed test suite may still satisfy the same testing requirements [3, 9]. Since the costs of executing test cases and managing test suites may often be quite significant, a small test suite that can still satisfy all testing requirements is desirable. This is known as test suite reduction problem, which is \(NP\)-complete [8].

Existing efforts for test suite reduction include the heuristic algorithms, 0-1 integral programming methods and so on [3, 8, 9]. Some case studies indicate that these methods perform well on the constructed test suites. The test suites are always reduced by analyzing the satisfiability relation between testing requirements and test cases. The test suite must be generated first before applying reduction techniques to a redundant set of test cases. Hence, they do not actually reduce the cost of test case generation. Moreover, there are always some complex interrelations among testing requirements. It is necessary and possible to optimize testing requirements before test case generation. It has more chances to generate a smaller test suite efficiently based on the smaller testing requirement set [11, 14].

Some works have been done to reduce the size of a test suite based on testing requirement analysis. Agrawal used the notion of dominators, super-blocks and mega-blocks to derive coverage implications among the basic blocks to reduce the coverage requirements for a program [1]. Similarly, Marre and Bertolino used a notion of entities subsumption to determine a reduced set of coverage entities such that coverage of the reduced set implies the coverage of the unreduced set [10]. These works exploit only the implications among the coverage requirements to generate a reduced set of coverage requirements. Tallam and Gupta explored the concept analysis framework in an attempt to derive a better heuristic for test suite minimization. Concept lattice exposed the implications among the test cases in a test suite as well as the implications among the requirements covered by the test suite [12]. Zhang et al. presented a testing requirement reduction model based on analyzing the interrelations among the testing requirements. Moreover, a testing requirement reduction method is proposed to generate the reduced testing requirement set [14]. In all the above approaches, testing requirement optimization is considered as a pre-process of test suite generation and reduction.

There are many software testing techniques in various fields of software engineering. Testing requirements defined
in different testing methods may be variant. In general, testing requirement optimization should be considered in preference to test suite reduction if it has the following properties. (1) The cost of test case generation is higher than testing requirement analysis. (2) The number of testing requirements is huge and it can be reduced potentially.

This paper studies the theoretical and practical results of testing requirement optimization for test suite reduction. Two main contributions of this paper are: (1) Testing requirement relation graph is presented to hide the details of testing requirements and test cases. Some requirement contraction methods are proposed to generate a small requirement set. (2) An empirical study on specification-based testing is performed. The study compares the relative effectiveness of several different methods based on variant measurements and priority orders for contraction.

2. TEST SUITE REDUCTION AND TESTING REQUIREMENT OPTIMIZATION

In this paper, we use $T = \{t_1, \cdots, t_n\}$ to denote a test suite and $R = \{r_1, \cdots, r_m\}$ to denote the set of testing requirements, in which each $r_i$ is considered as a subset of $T$. $\mathcal{P}(T)$ denotes the power set of $T$. A testing requirement is said to be feasible if there is a test case in the input domain that satisfies the testing requirement. We assume that each testing requirement in $R$ is feasible.

The relation between test cases and testing requirements can be represented as a bigraph $(T, R, E)$, in which $T$ and $R$ are two disjoint vertex sets and $E$ is a set of edges connecting $T$ and $R$. In this paper, we use a simple example of satisfiability relation $(T_0, R_0, E_0)$ to demonstrate test case reduction and testing requirement optimization, as shown in Figure 1.

The set of all requirements satisfied by $t \in T$ is defined as $\text{Req}(t) = \{r \in R | t \in r\}$. $\text{Req}(T') = \bigcup_{t \in T'} \text{Req}(t)$. A test suite $T'$ satisfies $R$ if for each testing requirement $r \in R$, there is at least one test case in $T'$ satisfying $r$, i.e., $\text{Req}(T') = R$. The set of all test suites satisfying $R$ is denoted by $TS(R)$. $T'$ is said to be a reduced test suite of $S$ if $T'$ is a subset of $T$ such that $T'$ can satisfy $R$, i.e., $T' \subseteq T$ and $T' \in TS(R)$. A reduced test suite $T'$ is said to be an optimal one of $S$ if for any reduced test suite $T'', |T'| \leq |T''|$. The optimal test suite may be not unique and they have the same size. The test suite reduction problem is to find an optimal test suite for a given test suite, which is equivalent to minimum set cover, a well-known NP-complete problem [2].

Given a test suite $T$ and two testing requirements $r, r' \in \mathcal{P}(T)$, $r$ and $r'$ are intersectant if $r \cap r' \neq \emptyset$, denoted by $r \bowtie r'$. $r$ is subsumed by $r'$ if $r \subseteq r'$, denoted by $r \preceq r'$. $r$ is said to be 1-1 redundant to $R$ if $r' \preceq r$ for some other $r' \in R$. Given two testing requirement sets $R, R' \subseteq \mathcal{P}(T)$, $R'$ is a contracted requirement set of $R$ if $TS(R') \subseteq TS(R)$. A contracted requirement set $R'$ is an optimal one of $R$ if for any contracted requirement set $R''$, $|R'| \leq |R''|$. $\text{OptRS}(R)$ denotes all optimal contracted requirement sets of $R$. The testing requirement optimization problem is to find an optimal contracted requirement set for a given testing requirement set.

The test suite reduction problem could be solved by testing requirement optimization. That is, for each optimal test suite $T'$, there exists one optimal contracted requirement set can generate $T'$ efficiently. The following section proposes some graph contraction methods to generate a small testing requirement set.

3. REQUIREMENT RELATION GRAPH CONTRACTION

In some cases, the relationship of testing requirements can be captured before the test case generation by requirement engineering, semantic analysis, program analysis, domain knowledge, testing history and etc. [10, 14]. A requirement relation graph is defined as follows.

**Definition 1 (Requirement Relation Graph).**

Let $R$ be a testing requirement set, a requirement relation graph of $R$, denoted by $G(R)$, is an undirected graph $(V, E)$, in which $V = R$ and $E = \{(r, r') | r \bowtie r'\}$.

Although $r \bowtie r'$ is defined based on the test cases, $r \bowtie r'$ used in the following context does not rely on test case generation. It can determine whether there exists one test case satisfying both $r$ and $r'$ by other analysis methods, and the detail information of test cases is not needed. Given a testing requirement set $R$, it can be mapped to a unique requirement relation graph $G(R)$. For example, the resulting requirement relation graph of $R_0$ is shown in Figure 2.

3.1 Requirement Contraction

In an undirected graph $G = (V, E)$, the degree of a vertex $v$ is the number of edges incident to $v$, denoted by $\text{deg}(v)$. $\text{ver}(G)$ denotes the set of vertices of $G$. Contractions are useful in situations where we wish to simplify a graph by identifying vertices that represent essentially equivalent entities. In graph theory, an edge contraction is an operation which removes an edge from a graph while simultaneously merging together the two vertices it used to connect. All other edges incident to either of the two vertices become incident to the single merged vertex. The requirement contraction needs a stronger condition.

**Definition 2 (Requirement Contraction).**

Let $G(R) = (V, E)$ be a requirement graph and $(u, v) \in E$. A contracted graph of $G$ on $(u, v)$ is defined as $G' = (V', E')$, in which (1) $V' = V \cup \{w\} - \{u, v\}$ ($w$ is a new requirement equivalent to $u \cap v$). (2) $E' = \{(x, y) \in E | x \notin \{u, v\} \cup \{w, x\} | x \bowtie x \in V' - \{w\}\}$.

Following by the above definition, a testing requirement set can be contracted one by one until there is no requirements can be contracted, i.e. each pair of requirements is
OptRS is a simplified graph of Figure 2. It is not difficult to see that OptRS(G(R), R) = OptRS(R). For a contracted graph G', OptRS(G', R) ⊆ OptRS(R). That is, some requirement sets may be destroyed in the process of requirement contraction. Hence OptRS(G', R) may be empty for some contracted graphs G'. A contracted graph G' is safe if OptRS(G', R) ≠ ∅.

A 1-1 contraction graph is a subgraph of G(R) by removing the 1-1 redundant requirements of R. It needs additional checking to determine 1-1 reduction requirements. The following theorem shows that 1-1 contraction is safe.

**Theorem 3 (1-1 Contraction).**

1-1 contraction is safe. That is, if r is a 1-1 redundant requirement of R, then

\[ \text{OptRS}(G_{R-r}, R) ≠ ∅ \]  

**Proof.** If r is 1-1 redundant to R, then the optimal contracted requirement set of R - {r} must be one of R, i.e. \( \emptyset ≠ \text{OptRS}(R - \{r\}) \subseteq \text{OptRS}(R) \). Since \( G(R - \{r\}) = G_{R-r} \), \( \text{OptRS}(G_{R-r}, R - \{r\}) = \text{OptRS}(R - \{r\}) \subseteq \text{OptRS}(R) \).

A vertex \( v \) is called a leaf vertex if it has a unity degree, i.e. \( \text{deg}(v) = 1 \). Let \( v \) be a leaf vertex in G and \( (v, v') \) be the edge incident to v. A contraction of G on \( (v, v') \) will split it into two parts: \( G_{v,v'} \) and \( G'_{v,v'} \), in which \( R' = R - \{v, v'\} \). Let \( LC(G, v) = G_{v,v'} U G'(v, v') \) and it is called a leaf contraction graph on vertex v. The following theorem shows that the leaf contraction graph is safe.

**Theorem 4 (Leaf Contraction).**

A leaf contraction graph is safe. That is, if v is a leaf vertex of G(R), then

\[ \text{OptRS}(LC(G, v), R) ≠ ∅ \]  

**Proof.** Since \( R = R' \cup \{v, v'\} \), a contracted requirement set of \( R' \cup \{v, v'\} \) is one of R. It only need to prove \( \forall R_1 \in \text{OptRS}(G_{v,v'}) \cap \text{OptRS}(\{v, v'\}), \forall R_2 \in \text{OptRS}(R) : |R_1| ≤ |R_2| \), that is, \( \forall R_3 \in \text{OptRS}(G_{R'}, R'|v,v' \in R_{R'} \cap \text{OptRS}(R) : |R_3| ≤ |R_2| - 1 \).

For \( R_2 \in \text{OptRS}(R) \), since \( v \) is independent to other vertices, except \( v' \), there exists one \( r' \subseteq R_2 : r' \subseteq v \) and \( r' \not\subseteq v'' \) for each \( v'' \subseteq R' \). Then \( R_2 - \{r'\} \) is a contracted requirement set of \( R' \). Hence the size of optimal one of \( R' \), \( |R_3| \), is not larger than \( |R_2 - \{r'\}| \), i.e. \( |R_3| ≤ |R_2| - 1 \). \( v \cap v' ≠ \emptyset \), because \( (v, v') \) is an edge of G. Hence

\[ \text{OptRS}(G_{v,v'}, R') \subseteq (v \cap v') \text{ is not empty.} \]

Figure 3 shows the contraction process of requirement relation graph of \( R_0 \). \( r_2 \) and \( r_3 \) is 1-1 redundant to \( r_5 \) and \( r_4 \), respectively. \( r_2 \) and \( r_3 \) can be removed safely and \( G_a \) is a simplified graph of Figure 2. \( r_3 \) is a leaf vertex in \( G_a \), then it can be simplified using leaf contraction. \( G_b \) is a contracted graph of \( G_a \). \( r_3 \) is a leaf vertex in \( G_a \), then it can be simplified using leaf contraction. \( G_b \) is a contracted graph of \( G_a \) and \( G_a \) is a contracted graph of \( G_b \).

**3.2 Priority for Contraction**

A challenge to requirement contraction is how to establish an order of priority to contraction, so that the resulting requirement set can be approximated to an optimal one. Given a requirement graph, if a subset of requirements can be contracted to one, then these requirements must belong to a same complete subgraph, i.e. each pair of distinct vertices in the subgraph are connected by an edge. Two simple measurements are proposed to replace maximal complete subgraph. For an edge \((u, v) \in E\), a simple measurement is the sum of degrees of vertices, i.e.

\[ D_1(u, v) = \text{deg}(u) + \text{deg}(v) \]

In some cases, the sum of degrees of \( u \) and \( v \) is large, but most edges of \( u \) and \( v \) are not adjacent. Hence they can not constitute a large complete subgraph. With the restriction of the edges of \( u \) and \( v \) must be connected, an other measurement is described as follows.

\[ D_2(u, v) = |\{x| (u, x) \in E \text{ and } (v, x) \in E\}| \]

Higher values of \( D_1 \) and \( D_2 \) may indicate a higher chance of a large complete subgraph, thus more testing requirements could be contracted to one potentially. Hence a higher value of measurement is preferential. On the other hand, a higher value of measurement indicates a higher chance of more complete subgraphs, i.e. more choices of contraction. Thus the proper choices are more difficult. Hence a lower value of measurement is preferential. The leaf contraction strategy is a good explanation of low value priority. These two opposite priorities should be compared by the experiments.

**4. PRACTICAL EXPERIENCE ON SPECIFICATION-BASED TESTING**
A Boolean expression is a well-defined string $s$ containing variables and the logic operations $\neg$, $\land$, $\lor$, etc. Given a Boolean specification $s$, an implementation expression may be a mutant $m$ by making simple syntactic changes to $s$. A test case $t$ is an assignment for all variables. $s(t)$ and $m(t)$ denote the values of $s$ and $m$ evaluated by the test case $t$, respectively. In general, a mutant $m$ may happen to be logically equivalent to the specification $s$ and hence it cannot be distinguished by any test case. A non-equivalent mutant is called a fault and an equivalent mutant is not regarded as a fault. A fault $m$ is said to be killed (or detected) by a test case $t$ if $m(t) \neq s(t)$. That is, $t$ is a satisfying assignment of $m \oplus s$ ($\oplus$ is the exclusive-or operator).

For each mutant $m_i$, a testing requirement is formed that $r_i = s \oplus m_i$. A testing requirement $r_i$ is feasible if and only if $r_i$ is satisfiable. For test suite reduction, all test cases, i.e., satisfying assignments, should be generated to satisfy each $r_i$. Then using some heuristic methods to reduce the test suite. Boolean specification-based testing can benefit from testing requirement optimization for the key points as follows.

1. Each testing requirement is formalized as a Boolean expression. The number of test cases is finite. For two testing requirements $r_i$ and $r_j$, $r_i \Leftrightarrow r_j$ if and only if $r_i \land r_j$ is satisfiable. $r_i \subseteq r_j$ if and only if $r_i \land \neg r_j$ is unsatisfiable.

2. In practice, generating all satisfying assignments is much harder than determining satisfiability. Although the satisfiability problem of a Boolean expression is $NP$-complete, many efficient SAT tools are developed and they can be used expediently.

3. The satisfiability problem of $r_i \land r_j$ would not be harder, actually easier in practice, than the one of $r_i$ and $r_j$. $r_i \land r_j$ is equivalently represented as the contracted requirement $r_i \cap r_j$.

### 4.1 Experiment Design

Our experimental analysis was done using software that was specifically designed and implemented for the purpose above. The software allows the analysis of a given Boolean specification. The 20 Boolean specifications were originated from the specification of an aircraft collision avoidance system called TCAS II [13]. The Boolean specifications were numbered 1 to 20, denoted by $Sp No$. in table 1. denotes The number of total test cases is $2^n$ if the number of variables is $n$.

Two of fault classes, LNF and LRF [5], was considered in the experiment. Literal Negation Fault (LNF): A literal is replaced by its negation, e.g., $(a \land b) \lor ( \neg b \land c)$ implemented as $(a \land b) \lor (b \land c)$ with $\neg b$ replaced by $b$. Literal Reference Fault (LRF): A literal is replaced by another literal that appears in the decision, e.g., $(a \land b) \lor (\neg b \land c)$ is implemented as $(a \land b) \lor (\neg a \land c)$ with $\neg b$ replaced by $\neg a$. For each literal in $s$, a mutant of LNF and mutants of LRF, $m_i$ were generated first, and then the testing requirements were formed as $s \oplus m_i$.

For each pair of testing requirements $r_i$ and $r_j$, we checked the satisfiability of $r_i \land r_j$ to determine the relation of $r_i$ and $r_j$. And then the requirement relation graph $G$ was constructed, in which a vertex was a testing requirement and an edge existed if and only if $r_i$ and $r_j$ were intersected.

For each pair of testing requirements $r_i$ and $r_j$ in $R$, we checked the satisfiability of $r_i \land \neg r_j$ and $\neg r_i \land r_j$ to determine the 1-1 redundant relation between $r_i$ and $r_j$. And then the 1-1 redundant requirements was removed. $1-1$ Con denotes the number of resulting testing requirements by 1-1 contraction.

For two measurements $D_1$ and $D_2$ described above, maximal and minimal measurement priorities were considered in the experiment. The sizes of resulting contracted requirement sets are denoted by $Max$ and $Min$, respectively. It is not difficult to see that leaf contraction is always preferential in the minimal measurement priority.

To compare with test suite reduction, all test cases of each testing requirement were generated first, although it was very expensive. And then the greedy algorithm was applied on the constructed test suites for reduction. $TSG$ denotes the size of resulting test suite.

### 4.2 Experimental Results and Analysis

The effectiveness of each approach was measured by computing the sizes of resulting requirement sets. Since the resulting requirement is independent, a minimal test suite can be constructed efficiently. Table 1 shows the experimental results on LNF and LRF. The bold number is the best result of all methods and Bes. denotes the number of occurrences of best results. To a further comparison, deviation analysis was introduced to quantify the goal of test suite reduction. The formalization is described as follows.

$$Dev. = \sum_{i=1}^{D} \frac{Result_i - BestResult_i}{BestResult_i}$$  \hspace{1cm} (5)$$

For a given method, higher values of Bes. and lower values of Dev. suggest that it obtains better results with respect to the other methods. Some observations are made as follows.

1. (Max vs. Min): The Min priority was found to be better than the Max priority. This could be concluded from the results of Bes. and Dev., wherever in $D_1$ or $D_2$, in LNF or LRF. The experimental results indicated that two requirements with less chances of contraction should be contracted preferentially. For this reason, leaf contraction should be implemented first.

2. (D1 vs. D2): The results of $D_1$ vs. the results of $D_2$ were very close. That is, varying measurements of testing requirements used on experiment had little effect on results. An interesting result on LRF was observed that the result of Bes. of $D_2$ was better and the result of Dev. of $D_1$ was better. Hence, $D_1$ and $D_2$ should be selected depending on different reduction targets.

3. (TS vs. RS): The results of test suite reduction (TS G) were better than the results of testing requirement optimization ($D_1$ and $D_2$). The greedy algorithm of test suite reduction selects the “most hitting” test case w.r.t. all requirements, which can be considered as a global operation. The contraction of testing requirement optimization focuses on two connected requirements, which can be considered as a local operation. A global operation is always better than a local operation, but it must weigh against the cost of global analysis, i.e. generation of all test cases.
5. CONCLUSION

In this paper, a graph contraction method is proposed to achieve test suite reduction. An empirical study on Boolean specification-based testing is performed to compare different methods of testing requirement optimization. The result of testing requirement optimization is no better than, but close to the result of test case reduction. However, the cost of generating all test cases is always very expensive. Testing requirement analysis is much cheaper in many cases. Testing requirement optimization can be implemented without the constructed test suite, thus it can reduce the cost of test case generation dramatically. On the other hand, the testing requirement contraction methods are still primary. There is a chance to improve testing requirement optimization.

6. REFERENCES