Coverage-Based Dynamic Mutant Subsumption Graph
Xiao-wei LI*, Ya-wen WANG and Huan LIN
No.10 Xitucheng Road, Haidian District, Beijing, China
*Corresponding author

Keywords: Coverage-based dynamic mutant subsumption graph, Mutation, Dynamic subsumption.

Abstract. Dynamic mutant subsumption graph (DMSG) provides a black-box definition for redundant mutants in finite set of tests. However, the graph is proved to be complicated and unreadable. In this paper we propose coverage-based subsumption graph, a novel model that describes the structure of DMSG based on coverage. That is: 1) mutants with the same coverage are composed of a local subsumption graph; 2) subgraphs of coverage are further connected together. In comparison with DMSG, this model is believed to be more readable and thus provides a coverage-based approach to generate the mutant subsumption efficiently.

Introduction

Mutation testing [1] is a test criterion that can be used to design test sets or evaluate test sets by generating a set of alternate programs, or mutants, and determining how many of these mutants are detected by the test suite. One long standing problem of using mutation testing is the cost of running too many mutants. Empirical observations on the Siemens suite [2], [3] showed that the number of redundant mutants, even when using selective mutation approaches, is quite large.

A promising approach to identify the redundancy in mutations is MSG (mutant subsumption graph) which naturally formed by subsumption relation. But “true” MSG is undecidable, so we may use dynamic mutant subsumption. Dynamic subsumption [4] provides a black-box approach to approximating the subsumption relationship based on the observed behavior of mutants with respect to a specific test set. Unfortunately, DMSG is unreadable and complicated.

This paper propose a new model, coverage-based dynamic mutant subsumption graph. First, we classify the mutants by coverage, the tests that execute the statement where mutants are seeded. The mutants with the same coverage constitute a “block”. Then we construct local DMSG for mutants in each block. In the end, subgraphs are further connected together by the subsumption relations between blocks. This coverage-based subsumption graph is more readable and provide approached to generate subsumption based on coverage information.

This paper is organized as follow. Section 2 develops the “true” subsumption relation and defines the “true” subsumption graph and dynamic subsumption graph. In section 3, we describe the model in detail. In section 4, there is an example to present our model. Section 5 presents related work. Section 6 discuss conclusion and future work.

Terms and Definition

“True” Subsumption

Even though “true” subsumption is uncomputable, it is useful to define it formally so as to provide a goal for the approximation approaches. In this paper, the term subsume without any modifiers means “true” subsumption.

Definition 1. $m_i$ subsumes $m_j$.

Let $M$ be a set of mutants on some artifact $A$. Denote an element of $M$ as $m$, possibly subscripted. We say that $m_i$ subsumes $m_j$, denoted $m_i \rightarrow m_j$, if and only if:

1) There exists some test $t$ such that $m_i$ and $A$ compute different outcomes on $t$ ($t$ kills $m_i$).
2) For every possible test \( t \) for \( A \), if \( m_i \) computes a different outcome than \( A \) on \( t \), then so does \( m_j \).

Three things are notable about the definition.

First, “true” subsumption considers the entire input domain of \( A \). That is, there are no limits on the test set.

Second, because of the first property, the definition does not allow vacuous subsumption. In other words, equivalent mutants are not part of the “true” subsumption relation. This is different from Jia and Harman’s definition of subsuming HOMs, which only requires the second property. That is, we require that we have a test that kills the subsuming mutant, whereas they do not. This matters because if a mutant subsumes another, but is equivalent to the original program (and thus cannot be killed), the subsumption relationship is irrelevant.

Third, the subsumption relation naturally forms a graph, which we call the Mutation Subsumption Graph or MSG. We defer the precise definition of the graph until we discuss some additional terms.

Let \( M_{\text{killable}} \) denote the set of non-equivalent mutants. A minimal set of mutants, \( M_{\text{minimal}} \) is a subset of \( M_{\text{killable}} \) with the following two properties:

1) Any test set that kills each mutant in \( M_{\text{minimal}} \) also kills each mutant in \( M_{\text{killable}} \).

2) If any mutant is removed from \( M_{\text{minimal}} \), the first property no longer holds.

Note that minimality for mutants is defined in terms of all possible test sets. Consider two mutants \( m_i \) and \( m_j \) in \( M \) that compute exactly the same function for all possible tests. We say that \( m_i \) and \( m_j \) are indistinguishable. From an analysis perspective, \( m_j \) adds no benefit beyond \( m_i \), and hence is redundant. With this observation, we are now ready to define the MSG:

1) Nodes in the MSG are maximal sets of indistinguishable mutants.

2) Edges in the MSG represent the subsumption relation.

Note that edges in the MSG only connect sets of mutants in \( M_{\text{killable}} \). All equivalent mutants occupy a single, isolated node in the graph.

If we ignore the equivalent mutants, then the remaining root nodes in the MSG capture everything needed to test artifact \( A \). In particular, if we restrict attention to \( M_{\text{killable}} \) and choose one mutant from each root node, then the resulting mutant set is minimal.

In drawing such graphs, it is common to show only the transitive reduction of the subsumption relation; otherwise the graphs quickly become unreadable.

**Dynamic Subsumption**

Subsumption relationships identify redundancy in sets of mutants and hence can be used to optimize approaches to both mutant and test generation. Specifically, subsumed mutants may not need to be generated, and test generation methods can target subsuming mutants. Thus, more knowledge about subsumption can help us build more efficient mutation testing tools, significantly improving the practical applicability of mutation in industry.

The problem with subsumption is the usual problem with general approaches to program analysis and testing: determining a subsumption relationship between two mutants is undecidable in general, and often challenging for cases where it can be computed. Hence, analysis approaches to subsumption can only offer partial answers.

**Definition 2.** \( S(T, x) \rightarrow S(T, y) \).

Let \( S(T, m) \) denote a vector whether each test case \( t \) kills mutant \( m \). So, if mutant \( x \) is not live and \( S(T, x) \in S(T, y) \), we say that \( m_x \) dynamically subsumes \( m_y \) with respect to \( T \), denoted \( S(T, x) \rightarrow S(T, y) \). In any set \( M \) that contains both \( m_x \) and \( m_y \), if \( m_x \) dynamically subsumes \( m_y \), then \( m_y \) is redundant, and hence may be safely discarded.

1. Difference between “true” subsumption and dynamic subsumption

This paper differentiates between two types of subsumption. True subsumption assumes full knowledge of the relationships among mutants and though valuable as a concept, is undecidable to compute, either through enumeration or analysis. Dynamic subsumption is computed relative to a specific set of tests. As the number of tests tends towards the entire domain of the artifact under test (not possible, of course), dynamic subsumption approaches “true” subsumption.
Model

In this paper we propose coverage-based subsumption graph, a novel model that describe the structure of DMSG based on coverage. Coverage of mutant specifies the tests that execute the statement where mutant is seeded. If mutant m1 and m2 are covered by the same tests, then m1 and m2 belongs to the same coverage. Mutants of the same coverage constitute a “block”. The subsumption graph for mutants in each block is called local MSG. Local MSG describes the subsumption between mutants with the same coverage.

Then, to construct the overall MSG, subsumption between mutants of different blocks need to be identified. Let B1 and B2 be the coverage of m1 and m2. There are four cases of the relationship between them:

1) \( B1 \cap B2 = \emptyset \);
2) \( B1 \cap B2 \neq \emptyset \) \& \( B1 \notin B2 \)
3) \( B1 \subseteq B2 \)
4) \( B1 = B2 \)

In case (1), there is no test case that can cover both mutants. In other words, m1 and m2 are located in contradicted paths. Since no test case can cover both mutants, there’s no test case that can kill both mutants. In other words, m1 is impossible to subsume m2, or the vice verse.

In case (2), mutant m1 subsumes m2, only when there is no test t in \( B1 - B2 \), such that t kills m1. If there’s such test, then such t cannot kill m2 because t does not execute the statement of m2. In other words, there’s test that kill m1 but cannot kill m2, and thus m1 does not subsume m2, which conflicts with the premise. Formally, for m1 to subsume m2 in case (2), following condition need to be met:

\[ B1 \cap (B1 - B2) = \emptyset \]

In static analysis, this condition can be used to validate the subsumption between two mutants. Similarly, in case (3), for mutant m2 to subsume m1, any tests in \( B2 - B1 \) must not kill m2. Otherwise, the subsumption from m2 to m1 is invalid.

In case (4), the constrain of coverage disappears. To prove whether m1 subsumes m2, only the tests in B1 is needed. This is because tests out of B1 do not kill m1 and influence on whether m1 subsumes m2. Similarly, in case (2) and (3), only tests in intersection \( B1 \cap B2 \) need to be investigated.

As we know, the graph is potentially huge and complicated, so we need to know what difference does the relationship between coverage make. The difference will help us construct DMSG more briefly and clearly.

Case (3) is the most valuable relationship which means that B1 subsumes B2. Only on this condition, can we construct DMSG. There will be an example in the next section.

To construct a coverage-based MSG, the following steps (Algorithm 1) will be performed:

1) Classify mutants based on their coverage in block.
2) Construct local MSG for mutants in each block. Note that, only the tests in coverage of the block need to be considered for identifying subsumption.
3) Build up the subsumption between mutants in different blocks.
4) Optimization based on four cases listed above will be applied to simplify the identification of inter-block subsumption.

**Algorithm 1 ConstructDMSG(B, M)**

Input: B - set of all block
Input: M - set of all mutants
Function: Mutate(x) – return the place where mutant x mutates
Output: DMSG - dynamic mutant subsumption graph

1. /* classify mutants based on their coverage in block */
2. for all mutants in M do
3. M[i].block = B[j]
4. end for
5. for m in M do
6. for b in B do
7. if Mutate(m) in b do
8. b.insert(m)
/* construct local MSG */
for B[i] in B do
    construtLMSG(i, B)
end for

/* build up subsumption between different block */
for all i, j in B and i != j do
    if i subsume j do
        Subsumption(i, j) = 1
    else
        Subsumption(i, j) = 0
    end for
end for

/* construct DMSG */
return genDMSG(B, M)

Case Study
In this section, we will list the result about a contrast experiments between DMSG and coverage-based DMSG and then show a case in detail.

The column Ori represent the time (ms) of every algoritms spent in original model, DMSG. The column Opt (ms) describes the time based in coverage-based DMSG.

In all algorithm, most execution time has been reduced about 0.12 except Insert Algorithm because the number of block in Insert Algorithm is too small to the result is almost same as traditional DMSG.

In detail description, we used the classical Prime Algorithm as our source code and mutation operators\cite{5} \cite{6} \cite{7} ProteumIM2.0 provided to generate mutants. In this case, we have generated 793 mutants and 14 test cases. Then coverage-based DMSG resulting from executing 14 test cases is shown in Figure 1.

Though analysis, the classical Prime Algorithm has four coverage denoted as B[0] to B[3]. The result of subsumption between every two coverage is in table 2 below. X and Y represent coverage number. Let Subsumption(x, y) = i (i ∈ 0, 1) denote that if X = x, Y = y, then Subsumption = i. For example, B[2] subsume B[0] and B[3] subsume B[2].

Table 1. Contrast.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Ori(ms)</th>
<th>Opt(ms)</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prime</td>
<td>12375</td>
<td>10519</td>
<td>0.15</td>
</tr>
<tr>
<td>Bubble</td>
<td>2120</td>
<td>1908</td>
<td>0.10</td>
</tr>
<tr>
<td>Calendar</td>
<td>95606</td>
<td>86177</td>
<td>0.09</td>
</tr>
<tr>
<td>Day</td>
<td>16258</td>
<td>13819</td>
<td>0.15</td>
</tr>
<tr>
<td>Insert</td>
<td>734</td>
<td>756</td>
<td>-0.03</td>
</tr>
<tr>
<td>Mid</td>
<td>1660</td>
<td>1428</td>
<td>0.14</td>
</tr>
<tr>
<td>Minmax</td>
<td>5623</td>
<td>4948</td>
<td>0.12</td>
</tr>
</tbody>
</table>

Table 2. Coverage Subsumption.

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Subsumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

362
In Figure 1, the four coverage denoted as $B[0]$ to $B[3]$, the same as above. After every identifier, the 14 bit binary code represent the condition whether the test case can cover the coverage or not.

In $B[0]$ there are 17 nodes, each node represents a set of one or more mutants that are Indistinguishable from each other, in the graph. What’s more, each dashed box represent an coverage.

**Definition 3.** $B[x]$ subsumes $B[y]$

If $B[x]$’s binary code is a true subset of $B[y]$’s binary code, then we denote $B[x]$ subsumes $B[y]$. Such as if the binary code of $B[x]$ is 001 and $B[y]$’s binary code is 011, then $B[x]$ subsumes $B[y]$ and $B[y]$ can not subsumes $B[x]$.

With the definition below, we can easily know that $B[1]$ subsumes $B[0]$, $B[2]$ subsumes $B[0]$ and $B[1]$, $B[3]$ subsumes $B[0]$, $B[1]$ and $B[2]$. In each coverage, there is a local MSG. For example, node 8 subsumes node 6, node 5 and node 7. So in process of generating DMSG, we can consider the mutants whose coverage has subsumption.

Although there are 793 mutants generated, but the coverage-based DMSG may also not contain all subsumption between mutants. With the number of test cases increases, the closer the coverage-based DMSG will be closer to “true” MSG. While the DMSG created from the larger combinatorial test set is closer to the “true” MSG, it may still be missing nodes and edges that would be produced by some even larger test set.

![Figure 1. Prime DMSG.](image)

**Related Work**

The subsumption relation has been studied in a variety of contexts for many years. Chusho observed that measuring branch coverage over all branches in a program led to an overestimation of quality, and defined the notion of essential branches as a way of removing redundant branches from coverage measures [8].

Jia and Harman defined the notion of subsuming Higher Order Mutants (HOMs) [9]. The idea was that a single HOM could stand in for several mutants. Langdon et al. applied subsuming HOMs to relational operators [10]. Lau and Yu presented a fault hierarchy of faults in Disjunctive Normal Form (DNF) predicates [11]. Kaminski et al. [12] extended this work by defining special HOMs, which, though relatively few in number, still subsumed all of the Lau and Yu hierarchy.

Kaminski et al. [13] observed that four of the seven mutants generated by Mothra’s Relational Operator Replacement (ROR) were always subsumed by other mutants. These work was done at operator level and it has a great impact on static subsumption.
Conclusion and Future Work

This paper proposes coverage-based DMSG. The model classifies mutants by coverage and makes it readable. It provides an efficient approach to construct the graph based on coverage.

With the limitations of experiment scale, there may have some problems and other more effective ways to generate the DMSG. If we can examine the model in real projects, maybe we can optimize the model.

Acknowledgment

The work described in this paper was supported by Key Project of the National Natural Science Foundation of China (No.61202080), National Natural Science Foundation of China (No.91318301), and China Postdoctoral Science Foundation (No.2015M581032).

Grateful acknowledgment is made to my Professor Mr. Gong who gave me considerable help by means of suggestion, comments and criticism. His encouragement and unwavering support has sustained me through frustration and depression. Without his pushing me ahead, the completion of this thesis would have been impossible. In addition, I deeply appreciate the contribution to this thesis made in various ways by my friends and colleagues.

Reference


