IMPROVING THE EFFECTIVENESS OF THE CLASSIFICATION-TREE METHODOLOGY IN SOFTWARE TESTING

T. Y. CHEN
Swinburne University of Technology
Australia

W. H. KWOK and T. H. TSE
The University of Hong Kong
Hong Kong

Abstract

This paper reports on the experience in the applications of the classification-tree methodology to a real-life system. As a result of the study, we found the need to supplement the original method by introducing a new relation operator and a new technique for constructing classification trees. Our supplement helps to enforce the completeness of test cases and improve on the effectiveness of classification trees.

Keywords: Black-box testing, classification trees, partition testing, software testing, specification-based testing, test case selection.

1 Introduction

As a black-box testing technique, the classification-tree method [1, 2, 3] was developed by Grochtmann and Grimm as an extension of the category-partition method [4, 5] for identifying classifications and their associated classes from a specification. They define classifications as the different criteria for partitioning the input domain of the program to be tested, and classes as the disjoint subsets of values for each classification. The classifications of the specification are arranged hierarchically to form a classification tree. Test cases for the specification are determined by joining the terminal classes of the tree in a systematic manner.

However, Grochtmann and Grimm assume that software testers can construct a classification tree from a specification according to their personal experience and expertise. No recommendation has been given on the actual process. Chen and Poon [6, 8] enhanced Grochtmann and Grimm's approach by proposing a methodology for building classification trees systematically. They introduced the concept of a classification-hierarchy table and a tree construction algorithm to handle the problems of ad hoc examination of the relations among classifications and joining classifications for building a classification tree. They defined four operators (\(\otimes\), \(\sim\), \(\Rightarrow\), and \(\Leftrightarrow\)) for categorizing the hierarchical relation of a pair of classifications. A summary of the operators is shown in Appendix A.

The algorithm of constructing a classification tree consists of (i) the construction of sub-trees associated with ancestor relations; (ii) the integration of the sub-trees by putting them under a root node; and (iii) the restructuring of duplicated classifications to improve the effectiveness of the classification tree.

It is possible to have test cases generated by a classification tree with incorrect class combinations. Each legitimate test case is formed by a set of classifications such that each classification only contributes one and only one of its classes to the test case. If a test case involves two mutually exclusive classes within the same classification, it will be illegitimate.

Let \(N_i\) be the number of legitimate test cases determined from a classification tree and \(N\) be the total number of test cases from the tree. The effectiveness of a classification tree is defined as the ratio of \(N_i\) to \(N\). In general, \(N_i \geq N\), and hence the effectiveness of a classification tree is less than or equal to 1.

To differentiate classifications and classes from normal text, we shall use the following two symbols throughout this paper: \(\Delta(X)\) indicates a classification \(X\) and \(\nabla(x)\) indicates a classification with incorrect class combinations.

2 Review of Original Construction Algorithm

2.1 Problem observed

We have studied the construction algorithm proposed by Chen and Poon by applying it to build a classification tree.
The classification hierarchy relations involved with the missing legitimate test cases are formed by combining the missing class combinations. In order that classifications described by coexistence relations will not be missed from a classification hierarchy table, we define \( \Delta(X) \) to be independent of \( \Delta(Y) \), denoted by \( \Delta(X) \odot \Delta(Y) \), if and only if any class \( V(x) \) in \( \Delta(X) \) can be part of some legitimate input irrespective of whether any class \( V(y) \) in \( \Delta(Y) \) is part of the same input. A characteristic of the relation is that given \( \Delta(X) \odot \Delta(Y) \), we have \( \Delta(Y) \odot \Delta(X) \).

Because of the introduction of the \( \odot \) operator, we need to modify the definition of the \( \ast \) operator as well. Thus, we define \( \Delta(A) \) to have other relations with \( \Delta(B) \), denoted by \( \Delta(A) \ast \Delta(B) \), if and only if \( \Delta(A) \) is neither a loose ancestor nor a strict ancestor nor independent of \( \Delta(B) \), and is not incompatible with it.

2.3 Problem resolution

To construct a classification tree covering all the valid class combinations according to Table 1, we should not only consider ancestor relations for grouping classifications into sub-trees. We must also group classifications with coexistence relations. In order that classifications described by coexistence relations will not be missed from a classification tree, we draw them as top-level classifications in the classification tree.

In this way, we have \( \Delta(\text{Exam Status}), \Delta(\text{Cancel Reason}) \) and \( \Delta(\text{User Reply}) \) as the top-level classifications of the new classification tree. The

We note from this table that all the problematic relations involve the \( \ast \) operator (referred to generally as "other relations" in the Chen and Poon method). On the other hand, the classification tree is derived only from relations involved with the \( \Rightarrow \) operator in the classification-hierarchy table (referred to as "ancestor relations"). The cause of the missing legitimate test cases is the exclusion of a special case of "other relations" when the classification tree is being constructed. We need to refine the "other relations" to tackle this problem.
Figure 1: Steps for Constructing a Catch-All Sub-Tree
coexistence relations in Table 2 suggest that we can arrange \( \Delta(Cancel\ Reason) \) and \( \Delta(Reply) \) as the children of every class of \( \Delta(Exam\ Status) \). This new arrangement leads to the classification tree of Figure 1(ii). Using the new classification tree, we have resolved the problem of missing legitimate test cases involving \( V(RG) \) of \( \Delta(Exam\ Status) \) and the classes of \( \Delta(Cancel\ Reason) \) and \( \Delta(Reply) \).

As for the classifications under \( V(AT) \) of \( \Delta(Exam\ Status) \), Table 2 reminds us to apply a similar arrangement to \( \Delta(Capture\ Mode) \). The resulting classifications are shown in Figure 1(iii).

Among the four classifications under \( V(AT) \), we have

\[ \Delta(Capture\ Mode) \Rightarrow \Delta(User\ Dialogue) \]

We have thus arranged \( \Delta(User\ Dialogue) \) as shown in Figure 1(iv), \( \Delta(User\ Dialogue) \) is an ancestor of \( \Delta(Cancel\ Reason) \) and \( \Delta(Reply) \) from Table 1. The resulting classifications are shown in Figure 1(v). The problem of missing legitimate test cases involving \( V(pending) \) of \( \Delta(Capture\ Mode) \) and the classes of \( \Delta(Cancel\ Reason) \) and \( \Delta(Reply) \) have been fixed by \( T'_1 \).

Through the construction of \( T'_1 \), we have illustrated the usefulness of the coexistence relation when constructing a classification tree. The legitimate test cases associated with the relations in Table 2 can be reflected in Figure 1(iii).

Furthermore, for this example, we need not apply the restructuring algorithm [7, 9, 10] to improve the effectiveness of \( T'_1 \) because the sub-tree \( S_{\Delta(User\ Dialogue)} \) appears only once in \( T'_1 \). As we illustrate below, restructuring is by no means a trivial process.

Referring to Figure 1(i), the arrangement of \( \Delta(Cancel\ Reason) \) and \( \Delta(Reply) \) under \( \Delta(User\ Dialogue) \) is the structure of \( S_{\Delta(User\ Dialogue)} \). Applying the restructuring algorithm to \( T_1 \), the sub-tree \( S_{\Delta(User\ Dialogue)} \) appeared under \( V(pe)/V(del) \) of \( \Delta(Report\ Status) \), \( V(PS) \) of \( \Delta(Link\ Type) \), and \( V(done) \) of \( \Delta(Capture\ Mode) \). Based on the criterion of returning the minimum number of test cases after restructuring, \( S_{\Delta(User\ Dialogue)} \) under \( V(PS) \) is retained.

For the manipulation of the classifications under \( V(AT) \), the only way to retain this sub-tree while maintaining the classes \( \Delta(Cancel\ Reason) \) and \( \Delta(Reply) \) with \( V(pending) \) of \( \Delta(Capture\ Mode) \) in \( T'_1 \) is to make use of the ancestor relation between \( \Delta(Capture\ Mode) \) and \( \Delta(User\ Dialogue) \) for establishing the hierarchy of these classifications.

3 Effectiveness Improvement with Catch-All Sub-Trees

In the previous section, we have demonstrated the use of both the coexistence relations and ancestor relations to construct a classification tree in a situation where the problems have already been identified. In general, we need to establish a systematic way for constructing a classification tree from a classification-hierarchy table based on the new concept. With this idea in mind, we recommend the following revised procedure:

(a) Construction of Catch-All Sub-Trees

Referring to Table 1 again, we note that top-level classifications cannot be descendants of other classifications. Furthermore, top-level classifications cannot be incompatible with other classifications either. However, any other classifications may be top-level classifications.

Based on these observations, we select the following as potential top-level classifications: classifications that can coexist with other classifications but are not descendants of any other classifications. We construct a sub-tree under each potential top-level classification.

In our earlier example, we have selected \( \Delta(Exam\ Status) \) as the only top-level classification of \( T'_1 \). In general, we recommend determining it in a systematic manner from the classification-hierarchy table. We consider every row of the classification table. If the classification corresponding to some row is a potential top-level classification, then construct a sub-tree with this classification as the root. In this case, construct other elements in the sub-tree based on every entry in the same row. We will refer to such a sub-tree as a catch-all sub-tree, since it captures all types of relations under a potential top-level classification, with the exception of "incompatible" relations. For example, \( T_1 \) is the catch-all sub-tree of the row in Table 1 corresponding to the potential top-level classification \( \Delta(Exam\ Status) \).

(b) Streamlining the Hierarchy of Catch-All Sub-Trees

Because of the coexistence relations among potential top-level classifications, we can establish a hierarchical relationship among the potential top-level classifications. This hierarchical relationship can be streamlined in a step-by-step fashion, as illustrated below. The streamlining steps enable us to ultimately select one of the potential top-level classifications as the only top-level classification of the final tree.

With Figure 1(v) as the starting point in determining the hierarchy of classifications under \( V(AT) \), we put \( \Delta(Link\ Type) \) as a child of every class of \( \Delta(Capture\ Mode) \). This is illustrated in Figure 1(vi). According to Table 1, \( \Delta(Link\ Type) \) is an ancestor of \( \Delta(User\ Dialogue) \), \( \Delta(Cancel\ Reason) \)
Table 3: Effectiveness Comparison of the Classification Trees

<table>
<thead>
<tr>
<th>Classification Tree</th>
<th>E_T</th>
<th>N_T</th>
<th>M_T</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_1</td>
<td>0.1754</td>
<td>57</td>
<td>10</td>
</tr>
<tr>
<td>T_2</td>
<td>0.1964</td>
<td>112</td>
<td>22</td>
</tr>
<tr>
<td>T_3</td>
<td>0.1754</td>
<td>57</td>
<td>42</td>
</tr>
</tbody>
</table>

4 Conclusion

In this paper, the construction of a classification tree based on the concept of catch-all sub-trees has been illustrated by a real-life example. A new operator ⊛ has been introduced in classification-hierarchy tables to highlight coexistence relations that are an important concept in the construction of classification trees.

Using the catch-all sub-tree construction approach, we have resolved the problem of missing valid class combinations caused by solely considering ancestor relations in the original construction algorithm. By applying the new approach to the same real-life example, we have also demonstrated a marked improvement in effectiveness of the resulting classification tree.

References


Appendixes

A Original Definitions Chen and Poon's Hierarchical Operators

(1) We define \( X \) to be a loose ancestor of \( Y \), denoted by \( X \prec Y \), if and only if the following conditions are satisfied:

(a) There exist some \( x \in X \) and \( y \in Y \) such that \( \text{class}(X) = x \) and \( \text{class}(Y) = y \) are part of a legitimate input.

(b) There exists some \( x' \in X \) such that, for any \( y' \in Y \), we cannot have \( \text{class}(X) = x' \) and \( \text{class}(Y) = y' \) in any legitimate input.

(c) There exists some \( y'' \in Y \) such that, for any \( X'' \in X \), we cannot have \( \text{class}(X) = x'' \) and \( \text{class}(Y) = y'' \) in any legitimate input.

(2) We define \( X \) to be a strict ancestor (or simply an ancestor) of \( Y \), denoted by \( X \prec\prec Y \), if and only if the following conditions are satisfied:

(a) There exist some \( x \in X \) and \( y \in Y \) such that \( \text{class}(X) = x \) and \( \text{class}(Y) = y \) are part of a legitimate input.

(b) There exists some \( x' \in X \) such that, for any \( y' \in Y \), we cannot have \( \text{class}(X) = x' \) and \( \text{class}(Y) = y' \) in any legitimate input.

(c) There does not exist any \( y'' \in Y \) such that, for any \( X'' \in X \), we cannot have \( \text{class}(X) = x'' \) and \( \text{class}(Y) = y'' \) in any legitimate input.

(3) We define \( X \) to be incompatible with \( Y \), denoted by \( X \sim Y \), if and only if for any \( x \in X \) and \( y \in Y \), we cannot have \( \text{class}(X) = x \) and \( \text{class}(Y) = y \) in any legitimate input.

(4) We define \( X \) to have other relations with \( Y \), denoted by \( X \not\prec\prec Y \), if and only if \( X \) is neither a loose ancestor nor a strict ancestor of \( Y \), and is not incompatible with it.

B Specification of Examination Cancellation

An active examination is classified into the types \( RG \) and \( AT \). The cancellation reason for an active examination is captured in a pop-up window. The cancellation action is committed by clicking the "OK" button of the window. The "Cancel" button of the window aborts the action. The cancellation reason is mandatory for committing the action.

(a) Examinations of \( RG \) Type

The captioned pop-up window is shown as above.

(b) Examinations of \( AT \) Type

Before the captioned pop-up window is shown, the following steps of checking are went through

\( (b_1) \) If the link type of an \( AT \) examination is \( PL \) or \( SU \), the examination cannot be cancelled. The possible link types for an \( AT \) examination are \( PS, PL \) or \( SU \).

\( (b_2) \) If the report status of an \( AT \) examination is \( pe \) or \( del \), the examination can be cancelled. The possible report status for an \( AT \) examination is \( ed \) (Edited), \( en \) (Endorsed), \( pe \) (Pending) and \( del \) (Deleted).

\( (b_3) \) Whether or not the data-capture work of an \( AT \) examination is completed, the examination can be cancelled. For \( AT \) examinations with completed data-capture work, a dialogue window is prompted to remind the status. Two options provided by the reminder are "Go" and "Stop". The cancellation action is halted after clicking the "Stop" button.

After going through the series of checking and satisfying the criteria as mentioned above, the cancellation reason window is shown.