Classification-tree restructuring methodologies: a new perspective

T.Y. Chen, P.-L. Poon and T.H. Tse

Abstract: The classification-tree method developed by Grochtmann et al. provided a useful approach for constructing test cases from functional specifications. It was automated by Chen and Poon through their tree construction methodology. In a follow-up study, Chen and Poon found that the effectiveness of constructing legitimate test cases could be improved under certain circumstances via a classification-tree restructuring algorithm. We develop another tree restructuring algorithm to cater for other situations not covered previously. The algorithms complement each other. We also compare the relative effectiveness of these algorithms and provide guidelines on applying them in practice.

1 Introduction

It is widely agreed [1–3] that, although software testing is an important component of the software development process, it is labour intensive and expensive and may account for 50% of the total cost of a project. Because of this high cost, testing should be as effective as possible. This goal can only be achieved if testing is well organised.

In software testing, a fundamental issue is the construction of a test suite, defined as a set of test cases that fulfills the testing requirement. The comprehensiveness of a test suite will affect the scope and quality of testing [4, 5]. In the past, numerous researchers have developed methods for constructing test suites from functional specifications (referred to as 'specifications' in this paper). In general, most of these test suite construction methods are based on formal specifications, such as Z specifications [6] or algebraic specifications [7, 8]. Among the few test suite construction methods that can be applied to informal specifications, a well known one is the category-partition method developed by Ostrand et al. [9, 10].

Using the category-partition method as a framework, Grochtmann et al. [11, 12] proposed a classification-tree method to help software testers to construct test suites. They define classifications as the criteria for partitioning the input domain of the program and classes as the disjoint subsets of values for each classification. Based on a specification, classifications and classes are arranged in the form of a hierarchical structure known as a classification tree, from which a test suite is generated [11]. Thus, the method adopts a black-box approach to the generation of test suites. However, it differs from other black-box methods in the following aspects: first, it constructs test cases primarily based on the information that is derived from the input domain (defined as the set of all possible inputs) of the program. Most of the other test suite construction methods focus on the functions provided by the program under test. Secondly, it can be applied to both formal and informal specifications, whereas most of the other test suite construction methods are only effective for formal specifications.

Despite the second advantage of the classification-tree method, it has a major drawback that hinders its widespread application. This drawback is that an ad hoc approach to the construction of classification trees has been proposed [11]. As a result, software testers with varying personal experience may construct different classification trees from the same specification. This inspired Chen and Poon [13] to develop a methodology for constructing a classification tree from a given set of classifications and associated classes via the notion of a classification-hierarchy table.

In general, a classification-hierarchy table captures the hierarchical relation between each pair of distinct classifications. An example of a hierarchical relation is that, when a classification Y takes a particular class x, classification Y must take a particular class y.

Chen and Poon [14] observed that:

(i) The quality of classification trees depends on the effectiveness of constructing legitimate test cases, defined as genuine cases that exist in the input domain and are therefore useful for testing. This quality is measured by the ratio of the number of legitimate test cases to the total number of test cases that can be constructed from the classification tree.

(ii) Classification trees of poor quality often have duplicated subtrees under different top-level classifications. These duplicated subtrees cause the classification tree to generate numerous illegitimate test cases, defined as test cases that do not exist in the input domain and are therefore not useful for testing.

From these observations, they defined an effectiveness metric to measure the quality of classification trees and developed a tree restructuring algorithm remove_duplicates to improve the value of the metric by removing duplicated
subtrees [14]. However, we have made a close examination of remove_duplicate and find that it is only effective for some types of classification trees. Specifically, our examination of remove_duplicate reveals that: first, the algorithm cannot handle subtrees that are duplicated within the same top-level classification. And, secondly, for classification trees with more than one set of duplicated subtrees under different top-level classifications, we can only select one of these sets to be handled by remove_duplicate. Furthermore, for this selected set \( S \), even if \( S \) contains more than two duplicated subtrees across different top-level classifications, remove_duplicate can only be applied once to two of these duplicated subtrees. In other words, this algorithm can neither be applied repeatedly for removing the remaining duplicated subtrees in \( S \), nor can the remaining sets of subtrees be duplicated elsewhere.

### 2 Previous work on the classification-tree method

#### 2.1 Original work

The classification-tree method [11, 12] was developed by Grochtmann et al. based on the category-partition method [9, 10] as the fundamental framework. It helps testers to generate test cases from specifications via the construction of classification trees. Basically, a classification tree organises the classifications and classes into a tree structure. The following describes the major steps of the method:

(i) Based on the specification, identify the major classifications of the program under test.

(ii) Identify the associated classes for every classification.

(iii) Organise all the identified classifications and classes into a classification tree.

(iv) Construct the test case table from the classification tree.

(v) Identify all possible combinations of classes from the test case table. Each of these combinations represents a potential test case.

(vi) Classify every potential test case as either a legitimate or illegitimate test case. The former should then be executed for testing the program, whereas the latter should be discarded.

We shall use Example 1 to illustrate the concept.

**Example 1 (selling of discounted tickets):** Supreme Airways has developed a program, known as ticket, to support the selling of air tickets to its staff at substantial discount rates. Two basic functions of ticket are to calculate the prices of the discounted tickets and the maximum weight of baggage that an employee can check in. The details of the specification for ticket are given in the Appendix (Section 8).

Suppose the classifications and classes for ticket are identified as in Table 1. As can be seen from the table, a class may correspond to a single value such as ‘rank of staff = supervisor’, or a range of values such as ‘mileage < 1000’. Because of the latter, given any classification \( X \), even though the union of all its classes should cover the part of the input domain applicable to \( X \), the number of classes for \( X \) is not necessarily large. Note also that, when there is no ambiguity, a class will simply be referred to as ‘supervisor’ or ‘<1000’ in this paper.

After identifying all the classifications and classes, an obvious approach is to select no more than one class from each classification so that each combination of selected classes forms a test case. For Table 1, for instance, a total of \( 3 \times 4 \times 3 \times 4 \times 3 \times 3 \times 3 \times 3 = 3888 \) test cases will be produced. (The number of possible ways to select no more than one class from any classification with \( n \) associated classes is \( n + 1 \) because none of these \( n \) classes may be selected.) However, some of these test cases are invalid because of the existence of incompatible classes. For example, according to clause (1) of the specification for ticket, the class ‘senior’ in the classification ‘seniority of staff’ cannot coexist with the class ‘clerk’ or ‘below’ in the classification ‘rank of staff’.

To reduce the number of invalid test cases, a classification tree is constructed. For instance, a classification tree for ticket, denoted by \( T_{\text{ticket}} \), is depicted in Fig. 1. The small circle at the top of the classification tree is the general root node, representing the whole input domain. The classifications directly under the general root node, such as ‘rank of staff’ and ‘type of ticket’ in Fig. 1, are called the top-level classifications.

In general, a classification \( X \) may have a number of classes \( x_i \) directly under it. \( X \) is known as the parent classification and each \( x_i \) is known as a child class. In Fig. 1, for example, ‘type of employment’ is the parent classification of ‘permanent’ and ‘temporary’, whereas ‘permanent’ and ‘temporary’ are the child classes of ‘type of employment’.

Similarly, a class \( x \) may have a number of classifications \( y_j \) directly under it. Then \( x \) is known as the parent class and each \( y_j \) is known as a child classification. In Fig. 1, for example, ‘domestic’ is the parent class of ‘mileage’, whereas ‘mileage’ is the child classification of ‘domestic’. We note that, in this particular figure, every parent class has only one child classification. In other classification trees, however, a parent class may have more than one child classification. An example is the parent class \( c_1 \) in Fig. 2, which has two child classifications \( D \) and \( E \).

Test cases can be expressed in a test case table under the classification tree. The columns of the test case table correspond to the terminal nodes of the classification tree and therefore represent classes. Each row corresponds to a combination of classes and therefore represents a potential test case. The test case table is constructed by the following steps:

(i) Construct the columns of the test case table by drawing a vertical line downwards from each terminal node of the classification tree.

(ii) Construct a test case in the test case table by selecting a combination of classes in the classification tree as follows:

<table>
<thead>
<tr>
<th>Table 1: Possible classifications and classes for ticket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classifications</td>
</tr>
<tr>
<td>Seniority of staff</td>
</tr>
<tr>
<td>Rank of staff</td>
</tr>
<tr>
<td>Type of employment</td>
</tr>
<tr>
<td>Type of airline</td>
</tr>
<tr>
<td>Class of ticket</td>
</tr>
<tr>
<td>Type of ticket</td>
</tr>
<tr>
<td>Mileage</td>
</tr>
</tbody>
</table>

1. Assume that ‘type of airline’ is ‘Supreme Airways’.
2. Assume that ‘class of ticket’ is ‘economy’.
3. Assume that ‘type of ticket’ is ‘domestic’.
4. Assume that ‘type of ticket’ is ‘overseas’.
5. Assume that ‘rank of staff’ is ‘manager or above’.
6. Assume that ‘seniority of staff’ is ‘senior’.
Fig. 1 Classification tree for 'ticket' and part of the test case table

(a) Select one and only one child class from each top-level classification.
(b) For every child classification of each selected class, recursively select one and only one child class.

For example, row 1 of the test case table in Fig. 1 represents the potential test case {manager or above, Supreme Airways, first, overseas}.

We use $P_i$ to denote a path in $T_{ticket}$. For example, $P_3$ denotes the path 'manager or above' -- 'Supreme Airways' -- 'economy'. Thus, the potential test case corresponding to row 1 is formed by selecting $P_1$ and $P_{10}$. Only part of the test case table is shown in Fig. 1. The complete table produces a total of 72 potential test cases. Compared to the 3888 test cases that would have been constructed by simply selecting and combining no more than one class from each classification in Table 1, we find that 3816 invalid test cases have been effectively filtered out. This elimination of invalid test cases has been achieved by capturing the hierarchical constraints among various classifications in $T_{ticket}$.

The classification-tree method has been used to test various real-life systems, such as a control system for the airfield lighting of an international airport [11], an identification system for automatic mail sorting machines [11], an integrated ship management system [11] and a simplified part of an adaptive cruise control system [15]. The results of these applications are very encouraging.

2.2 Subsequent work

Based on the original work of Grochtmann et al. on the classification-tree method, numerous related studies have been performed. In general, these studies aim at improving on the method by means of one of the following approaches:

---

IEE Proc.-Softw., Vol. 149, No. 2, April 2002
(i) Using more formal specifications.
(ii) Systematically capturing constraints among various classifications.

For approach (i), a well-known study is that performed by Singh et al. [15]. They developed a methodology for generating test cases from Z specifications based on the classification-tree method. They have applied the methodology to a simplified part of an adaptive cruise control system, which supports drivers by maintaining the speed of their vehicle at a safe distance from a preceding vehicle. Results show that the methodology can help software testers to identify classifications and classes from the specification.

Let us focus on approach (ii). Obviously, once the classification tree has been constructed, the formation of potential test cases is straightforward. Chen and Poon have noted, however, that the construction of classification trees as described in [11] is only ad hoc. It will be difficult, therefore, to apply the method when the specification is complex and involves a large number of classifications and classes.

This problem motivated Chen and Poon to develop a systematic tree-construction method via the notion of a classification-hierarchy table [13]. Basically, the table captures the hierarchical relation for every pair of distinct classifications. Once the table has been constructed, the corresponding classification tree can be formed using an associated tree construction algorithm.

Occasionally, a classification tree may not be able to reflect all the constraints among classifications. This problem results in the occurrence of illegitimate test cases. Hence, all the potential test cases constructed from the classification tree should be verified against the specification, with a view to identifying and removing all the illegitimate test cases before testing begins. For example, among the 72 potential test cases constructed from the classification tree T_ticket of Fig. 1, 57 are illegitimate. Only 15 potential test cases are legitimate and therefore useful for testing.

Chen and Poon [14] have proposed that the ultimate purpose of the classification-tree method is to construct legitimate test cases and the classification tree is merely a means for this construction. Given a classification tree T, let N(T, p) and N(T, l) be the number of potential test cases and legitimate test cases, respectively. Chen and Poon defined an effectiveness metric E[T] for T as follows:

\[ E[T] = \frac{N(T, l)}{N(T, p)} \]  

For example, since \( N(T_{\text{ticket}}, p) = 72 \) and \( N(T_{\text{ticket}}, l) = 15 \), \( E[T_{\text{ticket}}] \) is found to be 15/72 = 0.21. Obviously, \( N(T, l) \) can only be known after removing all illegitimate test cases from the set of potential test cases. On the other hand, even before the identification of individual potential test cases, \( N(T, p) \) can be derived directly from \( T \) using the formulae presented in [14].

Obviously, a small value of \( E[T] \) is undesirable because a considerable amount of effort will be wasted in the construction of illegitimate test cases. Furthermore, illegitimate test cases have to be identified manually from the set of potential test cases and hence human errors may result. If some legitimate test cases are mistakenly classified as illegitimate and consequently removed, the completeness of the set of legitimate test cases (and hence the comprehensiveness of testing) will be affected. Thus, it is highly desirable to develop ways of improving on the value of \( E[T] \).

Chen and Poon [14] proposed that a major cause for a poor \( E[T] \) is the existence of duplicated subtrees under different top-level classifications in a classification tree. Let \( S(X) \) denote a subtree with a classification \( X \) as its root and \( S(x) \) denote a subtree with a class \( x \) as its root. If \( X \) is a top-level classification, \( S(X) \) is called a top-level subtree. In \( T_{\text{ticket}} \) of Fig. 1, the subtree \( S(\text{type of airline}) \) is duplicated because it occurs in the top-level subtrees \( S(\text{rank of staff}) \) and \( S(\text{type of ticket}) \). As a result, we may construct an illegitimate test case containing the incompatible classes ('Supreme Airways' and 'other airline') by selecting either the set of paths \( \{P_1 \text{ or } P_2 \text{ or } P_3 \} \) or the set of paths \( \{P_4 \text{ or } P_10 \text{ or } P_11 \text{ or } P_{12} \} \). Similarly, since 'type of employment' is related to both the top-level classifications 'rank of staff' and 'type of ticket', the subtree \( S(\text{type of employment}) \) is also duplicated in both the top-level subtrees. These two duplications result in 57 illegitimate test cases, thereby reducing \( E[T_{\text{ticket}}] \) to a very small value.

From this observation, Chen and Poon developed a tree restructuring algorithm remove_duplicate to improve on the value of \( E[T] \) for classification trees with duplicated subtrees under different top-level classifications [14]. This improvement is achieved by removing the duplicated subtrees from the classification tree, thereby reducing the number of illegitimate test cases while preserving all the legitimate ones. For example, after applying remove_duplicate, the effectiveness metric of the classification tree for the bonus point program in [16] is increased from 0.17 to 0.40. Readers may refer to [14] for details.

The restructuring algorithm remove_duplicate may, however, convert some legitimate test cases into illegitimate ones through the introduction of incompatible classes. Hence, all the potential test cases constructed from the restructured classification tree must be reformatted using the algorithm described in [14]. The reformatting algorithm will ensure that any newly introduced illegitimate test cases are converted back into legitimate ones.

3 A new restructuring algorithm

Despite the ability to improve on the value of the effectiveness metric, we note two limitations in the restructuring algorithm remove_duplicate developed by Chen and Poon [14]: first, the algorithm assumes that classification trees do not have duplicated subtrees under the same top-level classification. Consider, for example, the classification tree \( T_{\text{ticket}} \) in Fig. 1. The subtree \( S(\text{class of ticket}) \) appears twice under the top-level classification 'rank of staff'. The algorithm cannot remove such duplications. Secondly, the algorithm can handle only one set of duplicated subtrees from the classification tree at any one time, even if the classification tree contains more than one set of duplications. For the set of duplicated subtrees \( S \) handled by remove_duplicate, even if \( S \) contains more than two duplicated subtrees across different top-level classifications, only two such duplicated subtrees would be handled by remove_duplicate. Furthermore, the follow-up reformatting algorithm will only work if remove_duplicate is run only once.

Consider, for example, the classification tree in Fig. 2. Suppose that, in this classification tree,

- \( r_1, r_2 \) and \( r_3 \) denote the top-level subtrees \( S(4), S(8) \) and \( S(9) \), respectively.
- \( S_{\text{h}}[C] \) and \( S_{\text{h}}[C] \) denote the subtrees within \( r_1 \) and \( r_2 \), respectively, with the classification \( C \) as their roots.
\* \( S_i[F] \), \( S_i[C] \), and \( S_i[X] \) denote the subtrees within \( \tau_i \) and \( \tau'_j \), respectively, with the classification \( F \) as their roots.

In this case, the algorithm can be used to restructure the classification tree by handling only one (but not both) of the following sets of duplicated subtrees:

(i) \( \{ S_i[C], S_i[C] \} \)

(ii) \( \{ S_i[F], S_i[F], S_i[F] \} \)

Furthermore, if applied to case (ii), the algorithm can only be used to handle any two (but not all) of the duplicated subtrees. Obviously, the effectiveness metric for the classification tree would be further improved if both sets could be handled, or all duplicated subtrees in any of these sets could be handled.

The above limitations motivated us to develop a new restructuring algorithm to supplement remove_duplicate. The new technique, known as remove_identical, is described as follows:

Let

- \( \tau_i, \ i = 1, 2, \ldots, w \) be the \( w \) top-level subtrees of a classification tree \( T \), where \( w \geq 2 \)
- \( S_i[X] \) be a subtree of \( \tau_i \) such that the root of \( S_i[X] \) is the classification \( X \)
- \( \tau'_j \) be the top-level subtree formed from \( \tau_j \) after pruning from it all the subtrees \( S_j[Z] \)

Suppose two or more of the top-level subtrees contain duplicated subtrees. Without loss of generality, let these top-level subtrees be \( \tau_1, \tau_2, \ldots, \tau_w \), where \( 2 \leq n \leq w \) and let the duplicated subtrees be \( S_1[X], S_2[X], \ldots, S_n[X] \) (Note that the number of duplicated subtrees may be more than the number of top-level subtrees with duplications. This is because the same subtree may occur more than once within a top-level subtree, such as \( T_{\text{dense}} \) in Fig. 1. The original algorithm remove_duplicate by Chen and Poon did not cater for this type of duplication and cannot, therefore, be used for restructuring such classification trees.) Suppose, further, that the situation is as follows:

- For every duplicated classification \( Y \) in \( T \), all the subtrees with classification \( Y \) as their roots are identical.
- Let classification \( U \) be a descendent of classification \( V \) in \( T \) (that is, \( U \) is placed under some class of \( V \) in \( T \)). For any class \( v \) of \( V \) if \( v \) can be combined with some class of \( U \) to form a legitimate test case, \( U \) will appear under this class \( v \).

Then:

Select a top-level subtree \( \tau_k \) (where \( 1 \leq k \leq n \)) such that, if we prune all the \( S_k[X], S_{k+1}[X], \ldots, S_n[X] \), \( S_k[X] \) from \( \tau_1, \tau_2, \ldots, \tau_{k+1}, \tau_{k+2}, \ldots, \tau_n \), respectively, it yields the smallest value of

\[
Q = \prod_{i=1}^{k-1} N(\tau_i) \times N(\tau_k) \times \prod_{j=k+1}^{n} N(\tau'_j)
\]

Replace the top-level subtrees \( \tau_1, \tau_2, \ldots, \tau_k-1, \tau_k+1, \ldots, \tau_n \) by \( \tau'_1, \tau'_2, \ldots, \tau'_k-1, \tau'_k+1, \ldots, \tau'_n \), respectively, but leave the selected top-level subtree \( \tau_k \) unchanged. If there are two or more distinct \( \tau_k \) that produce the smallest same value of \( Q \), then arbitrarily select any of them.

Repeat the above process until there are no duplicated subtrees \( S_i[X] \) and \( S_i[X] \) across any pair of distinct top-level subtrees \( \tau_i \) and \( \tau'_j \). Note, however, that \( S_i[X] \) is allowed to occur more than once within a top-level subtree \( \tau_i \). This completes the algorithm.

In the above algorithm, \( N(\tau_i) \) and \( N(\tau'_j) \) can be derived using the formulae from [14]. Suppose \( T' \) denotes the classification tree after restructuring. According to the formulae for the computation of \( N(T', p) \) in [14], the smaller the value of \( Q \), the smaller will be the value of \( N(T', p) \). Thus, by minimizing the value of \( Q \), we can improve on the value of \( E(T') \).

There are two important properties of remove_identical, as reflected in the two propositions that follow.

**Proposition 1 (convergence property):** Suppose a classification tree \( T \) has been restructured using the algorithm remove_identical to form \( T' \). The number of potential test cases constructed from \( T' \) will not exceed that from \( T \).

**Proof:** As seen from the restructuring algorithm remove_identical, \( T' \) is equivalent to \( T \) with some duplicated subtrees pruned. Obviously, the proposition follows immediately.

Before we proceed to prove the second property of the restructuring algorithm remove_identical, we have to introduce a few concepts. We define a feasible net, \( F \), in a classification tree as a collection of paths such that all the classes in each path form a potential test case. Thus, the number of distinct feasible nets in the classification tree is always equal to the number of potential test cases. For example, in the test case table of Fig. 1, the potential test case corresponding to row 2 is formed by selecting the feasible net that contains the paths \( \{ p_2, p_3 \} \).

Let \( \tau_i \) denote a top-level subtree in a classification tree. Given any feasible net \( F \) in the classification tree, a feasible subnet \( F_i \), is defined as the set of all feasible paths \( P \) in \( F \) such that all classes in each path of \( P \) is within \( \tau_i \). Suppose, for instance, \( \tau_i \) denotes the top-level subtree in Fig. 2 with classification \( X \) as its root. Then, \( F_i = \{ P_2, P_3 \} \) is a feasible subnet within \( \tau_i \).

Now, suppose a classification tree has two or more top-level classifications denoted by \( \tau_1, \tau_2, \ldots, \tau_w \). Suppose further that:

- \( \tau_i \) and \( \tau_j \) (where \( i \neq j \) and \( 1 \leq i, j \leq w \)) denote two distinct top-level subtrees containing duplicated subtrees \( S_i[X] \) and \( S_j[X] \), respectively.
- \( \tau_k \) (where \( k \neq i, k \neq j \) and \( 1 \leq k \leq w \)) denotes a top-level subtree that does not contain a duplicated subtree \( S_k[X] \).

The feasible subnets within \( \tau_i \) can be classified as follows:

(i) A feasible subnet \( F_i \), is in \( F(\tau_i, X) \) if some path in the subnet contains the classification \( X \).

(ii) A feasible subnet \( F_i \), is in \( F(\tau_i, \neg X) \) if no path in the subnet contains the classification \( X \).

Let us illustrate this concept using the classification tree in Fig. 2. Again, let \( \tau_i \) denote the top-level subtree \( S_i[A] \). Then, \( \{ P_2, P_3 \} \) is a feasible subnet in \( F(\tau_i, \neg C) \).

Having introduced the above concepts, we are now ready to prove the second property of the new restructuring algorithm remove_identical.

**Proposition 2 (preservation property):** Suppose a classification tree \( T \) has been restructured using remove_identical to form \( T' \). Any legitimate test case that can be constructed from \( T \) can also be constructed from \( T' \).

**Proof:** We shall follow the notation used in the restructuring algorithm remove_identical. Without loss of generality, let us assume that:

(a) The classification tree \( T \) has \( w \) top-level subtrees denoted by \( \tau_i, \ i = 1, 2, \ldots, w \), where \( w \geq 2 \).
(b) $\tau_1, \tau_2, \ldots, \tau_n$ (where $2 \leq n \leq w$) contain duplicated subtrees of the form $S_i[X], S_{i+1}[X], \ldots, S_n[X]$, respectively.  
(c) For any $1 \leq i \leq n$, $\tau_i$ denotes the top-level subtree formed by pruning all the subtrees of the form $S_i[X]$ from $\tau_i$.  
(d) After the application of remove_identical, all the duplicated subtrees in (b) are removed, except for the subtree(s) $S_i[X]$, in one and only one top-level subtree $\tau_k$ for some $1 \leq k \leq n$.  

Obviously, every feasible net $\mathcal{F}$ and the corresponding potential test case are formed by selecting one feasible subnet from every $\tau_i$, $i = 1, 2, \ldots, w$. Thus, any potential test case can be classified into one of the following types:  

(i) The potential test case is formed by selecting one feasible subnet from every $F(\tau_i, X)$, $i = 1, 2, \ldots, n$, and one feasible subnet from every $\tau_i$, $i = n + 1, n + 2, \ldots, w$.  

Obviously, all the feasible subtrees in $\tau_{n+1}, \tau_{n+2}, \ldots, \tau_w$ will remain intact after restructuring because every $\tau_{n+1} = \tau_{n+1}$, $\tau_{n+2} = \tau_{n+2}, \ldots$, and $\tau_w = \tau_w$.

Consider any feasible subnet $\mathcal{F}_{i_0}$ selected from $F(\tau_i, X)$ for some $i = 1, 2, \ldots, n$. Some path in $\mathcal{F}_{i_0}$, must contain some class within the duplicated subtree $S_i[X]$. Consider any such class $X$. It will obviously be deleted after pruning all the subtrees $S_i[X]$ from the classification tree $T$. Since $\tau_k = \tau_k$ for some $k = 1, 2, \ldots, n$, however, $X$ must still appear in some path of some feasible subnet in $F(\tau_i, X)$. Thus, all the potential test cases (and hence all the legitimate test cases) of this type can still be formed from $T'$.

(ii) The potential test case is formed by selecting feasible subtrees from a mixture of $F(\tau_i, X)$ and $F(\tau_j, \neg X)$ for $i, j = 1, 2, \ldots, n$ (where $i \neq j$), and one feasible subnet from every $\tau_i$, $i = n + 1, n + 2, \ldots, w$. We shall prove by contradiction that every potential test case of this type is illegitimate.

Suppose a potential test case of this type is legitimate. Consider any feasible subnet $\mathcal{F}_{i_0}$ selected from $F(\tau_i, \neg X)$. By definition, any path in $\mathcal{F}_{i_0}$ cannot contain the classification $Y$ and hence cannot contain any of its child classes. In other words, this path must contain some child class $y$ (in a classification $Y$) that cannot coexist with any child class in $X$. For any feasible subnet from $F(\tau_i, X)$, it must contain a child class $x$ in $X$. This will contradict the fact that the potential test case is legitimate. Hence, we need not be concerned whether this type of potential test cases originally constructed from $T$ will remain in $T'$ after restructuring.

(iii) The potential test case is formed by selecting one feasible subnet from every $F(\tau_i, \neg X)$, $i = 1, 2, \ldots, n$, and one feasible subnet from every $\tau_i$, $i = n + 1, n + 2, \ldots, w$. Such a test case, whether legitimate or otherwise, will remain unchanged after restructuring because every $F(\tau_i, \neg X)$ will be left intact and $\tau_{n+1} = \tau_{n+1}$, $\tau_{n+2} = \tau_{n+2}, \ldots$, and $\tau_w = \tau_w$.

Because of this, any legitimate test case that can be constructed from $T$ can also be constructed from $T'$.

Let us use example 2 to illustrate the application of the restructuring algorithm remove_identical and to show the improvement in $\mathcal{E}(T)$.

Example 2 (restructuring of classification trees): Consider the classification tree $T_{ticket}$ in Fig. 1. Let $\tau_1$ and $\tau_2$ denote the top-level subtrees $S_{[rank\ of\ staff]}$ and $S_{[type\ of\ ticket]}$, respectively.

Consider the duplicated subtrees $S_i_{[type\ of\ airline]}$ and $S_s_{[type\ of\ airline]}$. There are two alternate ways of restructuring $T_{ticket}$ using the algorithm remove_identical: (a) prune $S_i_{[type\ of\ airline]}$ from $\tau_1$ or (b) prune $S_s_{[type\ of\ airline]}$ from $\tau_2$.

Figs. 3 and 4 depict the two classification trees after the above ways of restructuring, respectively. Let $\tau_1$ be the result of pruning $S_i_{[type\ of\ airline]}$ from $\tau_1$ and $\tau_2$ be that of pruning $S_s_{[type\ of\ airline]}$ from $\tau_2$. Using the formulae presented in [14], $N(\tau_1) \times N(\tau_2) = 48$ for Fig. 3 and $N(\tau_1) \times N(\tau_2) = 45$ for Fig. 4. Hence, the restricted classification tree in Fig. 4 should be chosen.

A close examination of the restricted classification tree in Fig. 4 reveals that it still contains the duplicated subtrees $S_i_{[type\ of\ employment]}$ and $S_s_{[type\ of\ employment]}$. The restructuring algorithm remove_identical should therefore be applied again to remove duplications. The resultant classification tree $T'_{ticket}$ after the second application is depicted in Fig. 5.

From the preservation property of the restructuring algorithm remove_identical, we can guarantee that the 15 legitimate test cases constructed from the classification tree

---

**Fig. 3** Resultant classification tree $T'_{ticket}$ after pruning $S_i_{[type\ of\ airline]}$  

---

**Fig. 4** Resultant classification tree $T'_{ticket}$ after pruning $S_s_{[type\ of\ airline]}$  

---

**Fig. 5** Resultant classification tree $T'_{ticket}$ after the second application of the restructuring algorithm remove_identical.
4 Comparison of the two restructuring algorithms

Both the new restructuring algorithm remove_identical and Chen and Poon's previous algorithm remove_duplicate have merits. We would now like to compare them.

(a) With regard to the classification tree in Fig. 2, for instance, the new algorithm remove_identical is preferred to remove_duplicate. It is because both sets of duplicated subtrees $S_1 = \{S_1[C], S_1[C]\}$ and $S_2 = \{S_2[F], S_2[F], S_2[F]\}$ (note that $t_1, t_2, t_3$ denote the top-level subtrees with classifications $A, B$ and $G$ as their roots, respectively) can be handled by remove_identical, whereas only one of these sets can be handled by remove_duplicate. Furthermore, in case of $S_2$, remove_duplicate can only be used to remove one and only one of its duplicated subtrees.

(b) The new algorithm remove_identical can be applied to a classification tree with more than one duplicated subtree under the same top-level classification, such as $T_{ticket}$ in Fig. 1.

(c) The application of remove_identical does not require any follow-up by a reformatting algorithm.

(d) On the other hand, although remove_identical can be applied to a larger variety of classification trees with duplicated subtrees, remove_duplicate is more effective in improving on $E[T]$ in some situations, such as when all the following conditions are met:

(i) The classification tree contains only one set of duplicated subtrees denoted by $S$.

(ii) There are only two duplicated subtrees in $S$.

(iii) The two duplicated subtrees occur across different top-level classifications.

(iv) For every duplicated classification $X$ in the classification tree, all the subtrees with $X$ as their roots are identical.

We shall prove in proposition 3 that, under conditions (i) to (iv) in (d), remove_duplicate is more effective than remove_identical in improving $E[T]$. Before we do so, we shall use the following example to help readers to appreciate the relative effectiveness.
Example 3 (effectiveness): Consider the simple classification tree \( T_1 \) in Fig. 6. Using the formulae in \([14]\), we find that the total number of potential test cases \( N[T_1, p] \) is 12. We note that the structure of \( T_1 \) satisfies conditions (i)-(iv) in (d) above. It can be restructured by both remove_identical and remove_duplicate.

Fig. 7 depicts the classification tree \( T_1' \) after applying the algorithm remove_identical and Fig. 8 depicts the tree \( T_1'' \) after applying remove_duplicate. According to proposition 2, \( N[T_1', I] = N[T_1', I] \). According to \([14]\), \( M[T_1', I] = N[T_1', I] \). Hence, \( M[T_1', I] = N[T_1', I] \). Using the formulae in \([14]\), we find that \( N[T_1', p] = 8 \) and \( N[T_1', p] = 4 \). Since \( N[T_1', I] = N[T_1', I] \) and \( N[T_1', p] > => \( N[T_1', p] \), we have \( E[T_1'] < E[T_1''] \) by (1) quoted from \([14]\).

In other words, the previous algorithm remove_duplicate is more effective in this situation.

We are now ready to prove the following proposition:

Proposition 3 (effectiveness property): Given a classification tree \( T \), suppose

(i) \( T \) contains only one set of duplicated subtrees denoted by \( S \).

(ii) \( S \) contains only two duplicated subtrees.

(iii) The two duplicated subtrees occur across different top-level classifications.

(iv) For every duplicated classification \( X \) in \( T \), all the sub subtrees with \( X \) as their roots are identical.

Let

(a) \( T' \) denote the tree after applying the restructuring algorithm remove_identical.

(b) \( T'' \) denote the tree after applying the restructuring algorithm remove_duplicate instead.

Then \( N[T', p] \leq N[T'', p] \).

Proof: We shall follow the notation used in the restructuring algorithm remove_identical. Without loss of generality, suppose that

- \( T \) has \( w \) top-level subtrees denoted by \( t_i \), \( i = 1, 2, \ldots, w \), where \( w \geq 2 \).
- \( T \) contains only one set of duplicated subtrees \( \{S_m[X], S_n[X]\} \), where \( 1 \leq m, n \leq w \) and \( m \neq n \).
- \( S_m \) contains one and only one \( S_m[X] \).
- \( S_n \) contains one and only one \( S_n[X] \).
- For any duplicated classification \( Y \) in \( T \), every subtree with classification \( Y \) as its root is the same.

In this case, both remove_identical and remove_duplicate can be used to restructure \( T \).

For remove_identical, suppose

- \( S_m[X] \) is to be pruned from \( T \).
- \( S_m \) and \( S_n \) are restructured into \( T_m \) and \( T_n \), respectively.

Obviously, \( T_m = T_n \) and hence \( N(T_m) = N(T_n) \).

Consider remove_duplicate. Let

- The parent of \( X \) in \( T_m \) is the class \( z \) in the classification \( Z \).
- \( T_m \) and \( T_n \) are restructured into \( T_m' \) and \( T_n' \), respectively.

Furthermore, suppose the sequence of restructuring of the classification tree \( T \) is as follows:

1. The subtree \( A_1 \) is formed by pruning \( S_m[X] \) from \( S_n[Z] \).
2. The duplicated subtree \( S_m[X] \) is replaced by the subtree \( A_2 = Z - z - S_n[X] \).
3. \( S_n[Z] \) is replaced by a null tree, or \( A_3 \), or a modified \( A_3 \) with \( z \) deleted from it, depending on its initial structure before pruning \( S_n[X] \) from it in step (1).

It can be seen from step (2) that \( N(A_2) = N(S_n[X]) = N(S_n[Z]) \). Hence, \( N(T_m') = N(T_n') \). Since \( N(t_m') = N(t_n') \), we must have \( N(T_m') = N(T_n') \).

Now, consider the three cases for the replacement of \( S_n[Z] \) in step (3).

Case A: \( S_n[Z] \) is replaced by a null tree. This is equivalent to the pruning of \( S_n[Z] \) from \( T_m \). In this case, \( S_n[Z] \) must be the unique subtree of \( z \) and \( z \) must be the unique child class of \( Z \). Hence, the reduction of \( N(t_m') \) will be the same no matter whether we prune \( S_n[Z] \) as recommended by the restructuring algorithm remove_duplicate,
Thus, we must have $N_{\tau_m}(x) = N_{\tau_m}(\tau_m)$. Hence, we must have $N_{\tau_m}(\tau_m) = N_{\tau_m}(\tau_m)$.

Case C: $S_r[Z]$ is replaced by a modified $A_1$ with $x$ deleted from it. This is equivalent to the pruning of $S_r[Z]$ from $\tau_m$. In this case, $S_r[X]$ must be the unique subtree of $x$ and $z$ cannot be the unique child class of $z$. Otherwise, $S_r[Z]$ as recommended by remove_duplicated must be greater than that by pruning $S_r[Z]$ as recommended by remove_identical. Hence, we must have $N_{\tau_m}(z) = N_{\tau_m}(z)$.

Since $N_{\tau_m}(z) \leq N_{\tau_m}(x)$, $N_{\tau_m}(z) = N_{\tau_m}(\tau_m)$ and all the top-level subtrees, $s_i$ (where $1 \leq i \leq w$), $s_j \neq m$ and $t \neq n$ will remain intact after the application of remove_identical or remove_duplicate to $T$, we must have $N(T_{m,p}) \leq N(T_{m,p})$.

Proposition 3 indicates that remove_duplicate rather than remove_identical should be used as the restructuring mechanism when conditions (i)-(iv) in Section 4 are fulfilled. More specifically, the following guideline should be used for restructuring. Apply remove_duplicate if: (a) $T$ contains only one set of duplicated subtrees denoted by $S_r$ or (b) there are only two duplicated subtrees in $S$ or (c) the two duplicated subtrees occur across different top-level classifications rather than within the same top-level classification. Otherwise, apply remove_identical.

5 Conclusions

Testing plays an important role in verifying the correctness of software. As the quality of testing largely depends on the comprehensiveness of the test cases, it is essential to have a systematic method for constructing test cases from specifications. The classification-tree method developed by Grochtmann et al. [11] provides a useful direction. However, the construction of classification trees in their method is rather ad hoc, and hence a wide variation of trees may be constructed from the same specification according to the expertise and experience of the tester.

This problem was solved by Chen and Poon [13]. They provided a methodology for constructing a classification tree from a given set of classifications and associated class definitions via the notion of a classification-hierarchy table. The methodology further observed that: the quality of classification trees depends on the effectiveness of constructing legitimate test cases; and that a major reason for a poor quality is the occurrence of duplicated subtrees under different top-level classifications. From these observations, they first, defined an effectiveness metric for classification trees; and secondly, developed a tree restructuring algorithm remove_duplicate to improve on the value of the metric.

We have proposed in this paper a new restructuring algorithm remove_identical to supplement remove_duplicate. We have discussed the important properties of remove_identical. From these properties, we have provided guidelines to determine whether remove_duplicate or remove_identical should be used to reconstruct a given classification tree.

6 Acknowledgments

A preliminary version of this paper was presented at the Ninth International Workshop on Software Technology and Engineering Practice [16]. This work was supported in part by the Research Grants Council of Hong Kong under projects CityU 1118/99E and HKU 7029/01E.

7 References


8 Appendix

The following is the specification of the program ticket.

(1) Seniority and rank of staff

Every employee of Supreme Airways belongs to one of the following ranks:

- Rank 1: manager or above
- Rank 2: supervisor
- Rank 3: clerk or below

Employees of ranks 1 and 2 are referred to as 'senior' staff, whereas those of rank 3 are referred to as 'junior' staff.

(2) Types of employment

Every senior staff member is recruited on a permanent basis. On the other hand, a junior staff member may be recruited on either a permanent or a temporary basis.

(3) Types of airlines

Airlines are classified into two types, namely Supreme Airways and other associated airlines, because different discount rates apply.

(4) Classes of tickets

37
In general, (discounted) tickets are available in three different classes, namely 'first', 'business' and 'economy'.

(5) Types of tickets
Every (discounted) ticket can be classified into the type 'overseas' or 'domestic' depending on its destination.

Supreme Airways offers both types of discounted tickets to its staff. For all discounted tickets offered by Supreme Airways:

(a) If they belong to the type 'overseas', then their classes may be 'first', 'business' or 'economy'. Only senior staff are eligible to purchase this type of discounted tickets.
(b) If they belong to the type 'domestic', then their class must be 'economy'. All senior and junior staff are entitled to purchase this type of discounted tickets.

For all discounted tickets offered to Supreme Airways staff by any associated airlines, they must be of the type 'overseas' and the class 'economy'. Only staff of rank (1) are entitled to purchase this type of discounted tickets.

(6) Discounts
(a) For discounted tickets offered by Supreme Airways:
(i) If the employee is a senior staff member, then the discount rates for the classes 'first', 'business' and 'economy' are 60%, 70% and 80%, respectively.
(ii) If the employee is a junior staff member and is recruited on a permanent basis, then the discount rate is 80%.
(iii) If the employee is a junior staff member and is recruited on a temporary basis, then the discount rate is 75%.
(b) For discounted tickets offered by any associated airlines, the discount rate is 65%.

Because of clause 5 above:
- For all the discounted tickets in (6)(a)(i), their type may be 'overseas' or 'domestic'.
- For all the discounted tickets in (6)(a)(ii) and (6)(a)(iii), their type and class must be 'domestic' and 'economy', respectively.
- For all the discounted tickets in (6)(b), their type and class must be 'overseas' and 'economy', respectively.

(7) Maximum weights of baggage
(a) For discounted tickets offered by Supreme Airways:
(i) In the case of 'overseas' discounted tickets, the maximum weights of baggage for the classes 'first', 'business' and 'economy' are 40, 30 and 20 kg, respectively.
(ii) In the case of 'domestic' discounted tickets:
P. If the mileage is less than 1000 and the employee is a senior staff member, then the maximum weight of baggage is 15 kg.
P. If the mileage is less than 1000 and the employee is a junior staff member recruited on a permanent basis, then the maximum weight of baggage is 15 kg.
P. If the mileage is less than 1000 and the employee is a junior staff member recruited on a temporary basis, then the maximum weight of baggage is 10 kg.
P. If the mileage is not less than 1000, then the maximum weight of baggage is 20 kg for staff of all ranks.
(b) For discounted tickets offered by any associated airlines, the maximum weight of baggage is 20 kg.