Abstract

The classification-tree method developed by Grochtmann and Grimm facilitates the identification of test cases from functional specifications via the construction of classification trees. Their method has been enhanced by Chen and Poon through the classification-tree construction and restructuring methodologies. We find, however, that the restructuring algorithm by Chen and Poon is applicable only to certain types of classification trees. We introduce a new tree-restructuring algorithm to supplement their work.

1. Introduction

Motivated by the importance of test cases on the comprehensiveness and hence the quality of software testing [1, 2, 8, 9, 12, 14], numerous researchers have developed their own methods for constructing test cases from functional specifications (referred to as “specifications” in this paper). One of them is the classification-tree method [10, 11], which helps software testers construct test cases from specifications via the construction of classification trees. Although this method can be classified as a black-box testing approach, it differs from other black-box approaches in the following aspects:

(a) Most black-box testing techniques construct test cases from the functions of a program, whereas the classification-tree method achieves this from the input domain of a program. In other words, the former are basically function-oriented, whereas the latter is data-oriented [7].

(b) Most black-box testing techniques are only effective when the specification is written in a formal way, such
as in the formal specification language Z. On the other hand, the classification-tree method can be applied to both formal and informal specifications.

Since the tree construction approach proposed in [10, 11] is rather ad hoc, the classification trees constructed from the same specification may vary according to the personal experience of software testers. This inspired Chen and Poon [4, 6] 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. This table captures the hierarchical relation for each pair of distinct classifications. An example of a hierarchical relation is that, when a classification \( X \) takes a particular class, classification \( Y \) can take none of its classes. Furthermore, Chen and Poon observed that

(i) the quality of classification trees depends on the effectiveness of constructing legitimate test cases, and

(ii) a major reason for a poor quality in classification trees is the occurrence of duplicated subtrees under different top-level classifications.

For (ii), the duplicated subtrees cause the classification tree to generate numerous illegitimate test cases. As a result, the effectiveness with respect to legitimate test cases is reduced. From these observations, they defined an effectiveness metric to measure the quality of classification trees, and developed a tree-restructuring algorithm \textit{remove duplicate} for removing duplicated subtrees from classification trees, thereby improving on the value of the metric [5].

After a close examination, however, we find that \textit{remove duplicate} can only be applied to certain types of classification trees. There are cases where \textit{remove duplicate} cannot be applied. Specifically, our examination of \textit{remove duplicate} reveals that:

(a) The algorithm cannot handle subtrees that are duplicated within the same top-level classification.

(b) For classification trees with more than one set of duplicated subtrees under different top-level classifications, \textit{remove duplicate} can only be used to remove one of these sets at any one time. Furthermore, this algorithm cannot be applied repeatedly.

This paper addresses the above two issues. The rest of the paper is structured as follows. Section 2 reviews the previous work on the classification-tree method. Section 3 describes in detail our new restructuring algorithm. Finally, Section 4 concludes the paper.

2. Previous work on the classification-tree method

2.1 Grochtmann and Grimm

The classification-tree method [10, 11] was developed by Grochtmann and Grimm as an extension to the category-partition method [1, 3, 13, 14]. It helps testers construct test cases from specifications via the concept of classification trees. Classifications are defined as the criteria for partitioning the input domain of the program, whereas classes are defined as the disjoined subsets of values for each classification. Basically, a classification tree organizes the classifications and classes into a tree structure. The following describes the major steps of the classification-tree method:

1. Identify all the classifications and the associated classes from the specification.

2. Construct a classification tree from the classifications and classes.

3. Construct a test case table from the classification tree.

4. Identify all possible combinations of classes from the test case table. Each combination of classes represents a potential test case.

We shall use Example 1 to illustrate the concept.

Example 1

The software under test is the program \textit{bonus} being developed for Number-One Airline. It calculates the bonus points earned by passengers from their trips. Passengers can then claim various benefits such as free accommodation in leading hotels using the bonus points awarded.

The program calculates the bonus points according to the following specification:

1. **Classes of Seats**
   There are three classes of seats, namely first, business, and economy.

2. **Upgrading of Classes**
   Passengers holding an economy-class ticket are eligible for upgrading their tickets to a business class free-of-charge, provided that:
   
   (a) there are vacancies in the business class,
   
   (b) the passengers are holding a frequent-flyer card, and
   
   (c) the total mileage for the trip is less than 1000.
Under no circumstances can an economy-class or business-class ticket be upgraded to the first class.

(3) Discounts

Discounts are only available to:

(a) economy-class tickets, and
(b) the total mileage for the trip is not less than 1000.

There are two types of discounts, namely staff discount and passenger discount.

For (2)(c) and (3)(b), any distance less than one mile will not be counted. The number of bonus points earned will be calculated from the combination above. [The detailed calculations will be beyond the scope of this paper.]

Suppose the classifications and classes for bonus are identified as in Table 1. As seen from the table, a class may correspond to a single value such as “First”, or a range of values such as “≥ 1000”. Because of the latter, even though the union of all classes for any classification should cover the whole input domain, the number of classes for a single classification is not necessarily large.

After identifying all the classifications and classes, an obvious approach is to select a class from each classification so that each combination of selected classes forms a test case. For Table 1, for instance, a total of $3 \times 2^4 = 48$ test cases will be produced. However, some of these test cases are invalid because of the coexistence of incompatible classes. For example, according to clause (2) of the specification for bonus, the class “First” in the classification “Class of Seat” cannot coexist with the class “Yes” in the classification “Upgraded Class”.

In order to reduce the number of invalid test cases, a classification tree is constructed. For instance, a classification tree for bonus, denoted by $T_{bonus}$, is depicted in Figure 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 “Class of Seat” and “Total Mileage” in Figure 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 Figure 1, for example, “Price of Ticket” is the parent classification of “Normal” and “Discounted”, whereas “Normal” and “Discounted” are the child classes of “Price of Ticket”.

Similarly, a class $x$ may have a number of classifications $Y_j$ directly under it. Then $x$ is known as a parent class and each $Y_j$ is known as a child classification. In Figure 1, for example, “Discounted” is the parent class of “Type of Discount”, whereas “Type of Discount” is the child classification of “Discounted”.

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 represents a potential test case. The test case table is constructed by the following steps:

1. Draw the columns of the test case table by drawing a vertical line downward from each terminal node of the classification tree.
2. Construct a potential test case in the test case table by selecting a combination of classes in the classification tree as follows:
   (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 3 of the test case table in Figure 1 represents the potential test case such that:

- the “Class of Seat” is “Economy”,
- the “Price of Ticket” is “Discounted”,
- the “Type of Discount” is “Staff”, and
- the “Total Mileage” is “≥ 1000”.

We use $P_i$ to denote a path in $T_{bonus}$. For example, $P_2$ denotes the path “Class of Seat” – “Business” – “Upgraded Class” – “Yes”. Thus, the potential test case corresponding to row 3 is formed by selecting $P_3$ and $P_{10}$. Only part of the test case table is shown in Figure 1. The complete

<table>
<thead>
<tr>
<th>Classifications</th>
<th>Associated Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class of Seat</td>
<td>First, Business, Economy</td>
</tr>
<tr>
<td>Upgraded Class</td>
<td>Yes, No</td>
</tr>
<tr>
<td>Price of Ticket</td>
<td>Normal, Discounted</td>
</tr>
<tr>
<td>Type of Discount</td>
<td>Staff, Passenger</td>
</tr>
<tr>
<td>Total Mileage</td>
<td>&lt; 1000, ≥ 1000</td>
</tr>
</tbody>
</table>

Table 1. Possible classifications and classes for bonus
The classification-tree method has been used for testing various real-life systems, such as a control system for the airfield lighting of an international airport, an identification system for automatic mail sorting machines, and an integrated ship management system. The results of these applications are very encouraging [10, 11].

### 2.2 Chen and Poon

#### 2.2.1 Systematic construction of classification trees

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 [10, 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 [4, 6]. Basically, the table captures the hierarchical relation for every pair of distinct classifications. In the table for *bonus*, for example, we note that the classification “Type of Discount” cannot take any of its classes when the classification “Class of Seat” takes the class “First”. Once the classification-hierarchy table has been constructed, the corresponding classification tree can be formed using an associated tree construction algorithm. Readers may refer to [4, 6] for details.

#### 2.2.2 Restructuring of classification trees

Occasionally, a classification tree may not be able to reflect all the constraints among classifications. Hence, all the potential test cases constructed from the classification tree should be verified with the specification. In this way, we can identify and remove the potential test cases that contradict the specification. Such potential test cases are known as *illegitimate test cases*. Others are known as *legitimate test cases*. For example, row 2 of the test case table in Figure 1 will produce an illegitimate test case because, according to clause (2)(c) of the specification for *bonus*, the class “Yes” in the classification “Upgraded Class” cannot coexist with the class “$\geq 1000$” in the classification “Total Mileage”.

---

Figure 1. Classification tree for bonus point program and part of the test case table

The test case table produces a total of 30 potential test cases. Compared to the 48 test cases that would have been constructed by simply selecting and combining one class from each classification in Table 1, we find that 18 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 $T_{bonus}$. 

Note: Only part of the test case table is shown
In fact, 25 out of the 30 potential test cases constructed from $T_{\text{bonus}}$ are illegitimate. Only five potential test cases are legitimate and therefore useful for testing.

In [5], Chen and Poon 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 $N^p_T$ and $N^l_T$ 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:

$$E_T = \frac{N^l_T}{N^p_T}$$

For example, since $N^p_{T_{\text{bonus}}} = 30$ and $N^l_{T_{\text{bonus}}} = 5$, $E_{T_{\text{bonus}}}$ is found to be $\frac{5}{30} = 0.17$. Obviously, $N^l_T$ can only be known after removing all the illegitimate test cases from the set of potential test cases. On the other hand, even before the identification of potential test cases, $N^p_T$ can be derived directly from $T$ using the formulae presented in [5].

Obviously, a small value of $E_T$ is undesirable, as effort will be wasted on illegitimate test cases. In [5], Chen and Poon observed that a major cause of 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, we will call $S[X]$ a top-level subtree.

In $T_{\text{bonus}}$ of Figure 1, since “Price of Ticket” is related to both the top-level classifications “Class of Seat” and “Total Mileage”, the subtree $S[\text{Price of Ticket}]$ is duplicated in both the top-level subtrees $S[\text{Class of Seat}]$ and $S[\text{Total Mileage}]$. As a result, it is possible to construct an illegitimate test case containing the incompatible classes

“(“$<$ 1000” and “Normal”)

or (“$<$ 1000” and “Discounted”)

by selecting

$$(P_1 \lor P_3) \land (P_4 \lor P_5 \lor P_6).$$

Similarly, since “Upgraded Class” is related to both the top-level classifications “Class of Seat” and “Total Mileage”, the subtree $S[\text{Upgraded Class}]$ is also duplicated in both the top-level subtrees. These two duplications result in 25 illegitimate test cases, thereby reducing $E_{T_{\text{bonus}}}$ to a very small value.

From this observation, Chen and Poon developed a tree-restructuring algorithm $\text{remove\_duplicate}$ to improve on the value of $E_T$ for classification trees with duplicated subtrees under different top-level classifications [5]. 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, $E_{T_{\text{bonus}}}$ can be increased from 0.17 to 0.40 after the application of $\text{remove\_duplicate}$.

The restructuring algorithm $\text{remove\_duplicate}$ 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 [5]. The reformattting 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 $E_T$, we observe that $\text{remove\_duplicate}$ has two limitations:

(a) The algorithm only deals with the removal of duplicated subtrees under different top-level classifications. Consider, for example, the classification tree in Figure 2. The subtree $S[C]$ with classification $C$ as its root appears twice under the top-level classification $B$. The algorithm cannot remove such duplications.

(b) The algorithm can remove 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. Furthermore, the follow-up reformatting algorithm will only work if $\text{remove\_duplicate}$ is run only once. Suppose that, in Figure 1,
• $\tau_1$ and $\tau_2$ denote the top-level subtrees $S[\text{Class of Seat}]$ and $S[\text{Total Mileage}]$, respectively.
• $S_{\tau_1}[\text{Upgraded Class}]$ and $S_{\tau_2}[\text{Upgraded Class}]$ denote the subtrees $S[\text{Upgraded Class}]$ within $\tau_1$ and $\tau_2$, respectively, and
• $S_{\tau_1}[\text{Price of Ticket}]$ and $S_{\tau_2}[\text{Price of Ticket}]$ denote the subtrees $S[\text{Price of Ticket}]$ within $\tau_1$ and $\tau_2$, respectively.

In this case, the algorithm can be used to restructure $\mathcal{D}_{\text{bonus}}$ by removing only one (but not both) of the following sets of duplicated subtrees:

- $\{S_{\tau_1}[\text{Upgraded Class}], S_{\tau_2}[\text{Upgraded Class}]\}$
- $\{S_{\tau_1}[\text{Price of Ticket}], S_{\tau_2}[\text{Price of Ticket}]\}$

The improvement in $E_{\mathcal{D}_{\text{bonus}}}$ would of course be larger if both of the above sets could be removed.

The above limitations motivated us to develop a new restructuring algorithm for improving on the value of $E_{\mathcal{T}}$. The new algorithm, known as remove_identical, is described as follows:

**Tree Restructuring Algorithm remove_identical for a Classification Tree with Duplicated Subtrees:**

Suppose

(a) the classification tree has $w$ top-level subtrees denoted by $\tau_i$, $i = 1, 2, \ldots, w$, where $w \geq 2$.
(b) $N(\tau_i)$ denotes the total number of combinations of classes for $\tau_i$.
(c) $S_{\tau_i}[X]$ denotes a subtree of $\tau_i$ such that the root of $S_{\tau_i}[X]$ is the classification $X$.
(d) $\tau'_i$ denotes the top-level subtree formed from $\tau_i$ after pruning from it all the subtrees $S_{\tau_i}[X]$, and
(e) $N(\tau'_i)$ denotes the total number of combinations of classes for $\tau'_i$.

Suppose there are two or more top-level subtrees containing duplicated subtrees. Without loss of generality, let these top-level subtrees be $\tau_1, \tau_2, \ldots, \tau_n$, where $n \geq 2$, and let the duplicated subtrees be $S_{\tau_1}[X], S_{\tau_2}[X], \ldots, S_{\tau_n}[X]$.\(^a\)

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

$$Q = \left( \prod_{j=1}^{k-1} N(\tau'_j) \right) \times N(\tau_k) \times \left( \prod_{j=k+1}^{n} N(\tau'_j) \right)$$

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. In case there are two or more distinct $\tau_k$ that produce the same smallest value of $Q$, then arbitrarily select any of them.

Repeat the above process until there are no duplicated subtrees $S_{\tau_j}[X]$ and $S_{\tau_k}[X]$ across any pair of distinct top-level subtrees $\tau_j$ and $\tau_k$. Note, however, that $S_{\tau_j}[X]$ is allowed to occur more than once within a top-level subtree $\tau_k$.\(^a\)

In the above algorithm, $N(\tau'_j)$ and $N(\tau_k)$ can be derived using the formulae from [5]. Suppose $\mathcal{T}'$ denotes the classification tree after restructuring. According to the formulae for the computation of $N^P_{\mathcal{T}'}$ in [5], the smaller the value of $Q$, the smaller the value of $N^P_{\mathcal{T}'}$. Thus, by minimizing the value of $Q$, we can improve on the value of $E_{\mathcal{T}'}$.

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

**Proposition 1 (Convergence Property)**

Suppose a classification tree $\mathcal{T}$ has been restructured using the algorithm remove_identical to form $\mathcal{T}'$. The number of potential test cases constructed from $\mathcal{T}'$ will be no more than that from $\mathcal{T}$.

**Proof**

As seen from the restructuring algorithm remove_identical, $\mathcal{T}'$ is equivalent to $\mathcal{T}$ with some duplicated subtrees pruned. Obviously, the proposition follows immediately. \(\blacksquare\)

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 $\mathcal{F}$ in a classification tree as a collection of paths such that all the classes in these paths are selected together to form a whole 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 Figure 1, the potential test case corresponding to row 1 is formed by selecting the feasible net that contains the paths $P_1$ and $P_3$. In most cases, a feasible net will contain more than one path because:

---

\(^a\)We note 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. 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.
(a) a typical classification tree contains more than one top-
level subtree, and

(b) at least one path within each top-level subtree must be
selected to form a potential test case.

Let $\tau_i$ denote a top-level subtree in a classification tree. Given any feasible net $\mathcal{F}$ in the classification tree, a feasible subnet $\mathcal{F} |_{\tau_i}$ is defined as the set of all paths $\mathcal{P}$ in $\mathcal{F}$ such that $\mathcal{P}$ is within $\tau_i$. For example, suppose $\tau_1$ denotes the top-
level subtree in Figure 2 with classification $F$. Then, $\{P_1, P_3\}$ is a feasible subnet within $\tau_1$. It is obvious that:

(i) If every class in a classification tree has only one child
classification, as in Figure 1, then all the feasible sub-

nets in the tree will contain only one path.

(ii) If some class in a classification tree has two or more
child classifications, as in Figure 2, then some of the
feasible subnets in the tree will contain more than one
path.

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

(a) $\tau_i$ and $\tau_j$ (where $1 \leq i, j \leq w$) denote two distinct top-
level subtrees containing duplicated subtrees $S_{\tau_i}[X]$ and $S_{\tau_j}[X]$, respectively, and

(b) $\tau_k$ (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
$S_{\tau_i}[X]$.

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

(i) A feasible subnet $\mathcal{F} |_{\tau_i}$ is in $F(\tau_i, X)$ if some path in
the subnet contains the classification $X$.

(ii) A feasible subnet $\mathcal{F} |_{\tau_i}$ is in $F(\tau_i, \neg X)$ if no path in
the subnet contains the classification $X$.

Let us illustrate this concept with the classification tree
in Figure 2. Again, let $\tau_1$ denote the top-level subtree $S[A]$. Then, $\{P_1, P_3\}$ is a feasible subnet in $F(\tau_1, C)$, and $\{P_3\}$ is a feasible subnet in $F(\tau_1, \neg C)$.

Having introduced the above concepts, we are now ready
to prove the second property of the new restructuring algo-

Proving Proposition 2 (Preservation Property)
Suppose a classification tree $T$ has been restructured using $\text{remove_identical}$ 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 algo-
rithm $\text{remove_identical}$. Without loss of generality, suppose 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 $n \leq w$) contain duplicated subtrees
of the form $S_{\tau_i}[X], S_{\tau_j}[X], \ldots, S_{\tau_n}[X]$, 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 $S_{\tau_i}[X]$ from $\tau_i$, and

(d) after the application of $\text{remove_identical}$, all the dupli-
cated subtrees in (b) are removed, except for the sub-
tree(s) $S_{\tau_i}[X]$ 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 correspon-
ding potential test case are formed by selecting one feasible
subnet from every $\tau_i, i = 1, 2, \ldots, w$. Any legitimate test case
constructed from $T$ can be classified into two types:

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

(ii) The legitimate test case is formed by selecting one fea-
sible 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 will remain unchanged after restruc-
turing because:

- every $F(\tau_i, \neg X)$ will be left intact, and
- $\tau'_1 = \tau_{n+1}, \tau'_2 = \tau_{n+2}, \ldots, \tau'_w = \tau_w$.

We shall prove by contradiction that no third type of legiti-
mate test case exists.
Suppose a legitimate 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 subtree from every \(\tau_i, i = n + 1, n + 2, \ldots, w\). Consider any feasible subtree \(F|_{\tau_j}\) selected from \(F(\tau_i, \neg X)\). By definition, any path in \(F|_{\tau_j}\) cannot contain the classification \(X\), 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 test case is legitimate.

Thus, 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 \(E_T\).

**Example 2**
Consider the classification tree \(T'_{\text{bonus}}\) in Figure 1. Let \(\tau_1\) and \(\tau_2\) denote the top-level subtrees \(S[\text{Class of Seat}]\) and \(S[\text{Total Mileage}]\), respectively.

Consider the duplicated subtrees \(S_{\tau_1}[\text{Price of Ticket}]\) and \(S_{\tau_2}[\text{Price of Ticket}]\). There are two alternative ways of restructuring \(T'_{\text{bonus}}\) using the algorithm remove_identical:

1. Prune \(S_{\tau_1}[\text{Price of Ticket}]\) from \(\tau_1\), or
2. Prune \(S_{\tau_2}[\text{Price of Ticket}]\) from \(\tau_2\).

Figures 3 and 4 depict the two classification trees after the above ways of restructuring, respectively. Let \(\tau_1'\) be the result of pruning \(S_{\tau_1}[\text{Price of Ticket}]\) from \(\tau_1\), and \(\tau_2'\) that of pruning \(S_{\tau_2}[\text{Price of Ticket}]\) from \(\tau_2\). Using the formulae presented in [5], \(N(\tau_1') \times N(\tau_2') = 20\) for Figure 3, and \(N(\tau_1') \times N(\tau_2') = 18\) for Figure 4. Hence, the restructured classification tree in Figure 4 should be chosen.

A close examination of the restructured classification tree in Figure 4 reveals that it still contains the duplicated subtrees \(S_{\tau_1}[\text{Upgraded Class}]\) and \(S_{\tau_2}[\text{Upgraded Class}]\). The restructuring algorithm remove_identical should therefore be applied again to prevent duplication. The resultant classification tree \(T''_{\text{bonus}}\) after the second application is depicted in Figure 5.

From the preservation property of the restructuring algorithm remove_identical, we can guarantee that the five legitimate test cases constructed from \(T''_{\text{bonus}}\) before restructuring can still be constructed from \(T''_{\text{bonus}}\). Hence, \(N_{T''_{\text{bonus}}} = N_{T''_{\text{bonus}}} = 5\). On the other hand, \(N_{T''_{\text{bonus}}} = 12\) using the formulae in [5]. Thus, \(E_{\text{bonus}}^{T''_{\text{bonus}}} = \frac{12}{12} = 0.42\). When compared with \(E_{\text{bonus}}^{T''_{\text{bonus}}} = 0.17\), the improvement is about 147% and therefore quite significant.

**4. Conclusion**

Chen and Poon [4, 6] provided a methodology for constructing a classification tree from a given set of classifications and associated classes via the notion of a classification-hierarchy table. They observed that (a) the quality of classification trees depends on the effectiveness of constructing legitimate test cases, and (b) a major reason for a poor quality is the occurrence of duplicated subtrees under different top-level classifications.
these observations, they defined an effectiveness metric for classification trees and 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 proved that our new algorithm not only preserves the legitimate test cases but is also converging.

References


