Postprint of article in *IEE Proceedings: Software* **149** (2): 65–74 (2002)

Classification-Tree Restructuring Methodologies: a New Perspective [*][†]

T. Y. Chen [‡], Pak-Lok 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. In this paper, we develop another tree restructuring algorithm to cater for other situations not covered previously. The two algorithms will complement each other. We also compare the relative effectiveness between these algorithms, and provide guidelines on how to apply them in practice.

1 Introduction

It is widely agreed [2, 3, 4] 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 planned and organized.

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 the quality of testing [5, 6]. 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 [7] or algebraic specifications [8, 9]. 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. [10, 11].

Using the category-partition method as a framework, Grochtmann et al. [12, 13] proposed a classification-tree method to help software testers to construct test suites. They define *classifications* as the criteria for partitioning

[*] © 2002 IEE. This material is presented to ensure timely dissemination of scholarly and technical work. Personal use of this material is permitted. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder. Permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEE.

[†] A preliminary version [1] of this paper was presented at the 9th International Workshop on Software Technology and Engineering Practice, 1999.

[‡] T. Y. Chen is with the School of Information Technology, Swinburne University of Technology, Hawthorn 3122, Australia. E-mail: tychen@it.swin.edu.au

[§] Pak-Lok Poon is with the Department of Accountancy, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong. E-mail: acplpoon@inet.polyu.edu.hk

[¶] **Contact author.** T. H. Tse is with the Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong. E-mail: thtse@cs.hku.hk

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 [12]. 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:

(a) 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. On the other hand, most of the other test suite construction methods focus on the functions provided by the program under test.

(b) 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 advantage of the classification-tree method as mentioned in (b), the method has a major drawback that hinders its widespread application. This drawback is that an *ad hoc* approach to the construction of classification trees is proposed in [12]. As a result, software testers with varying personal experience may construct different classification trees from the same specification. This inspired Chen and Poon [14] 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 X takes a particular class x, classification Y must take a particular class y.

Chen and Poon [15] 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_duplicate* for improving the value of the metric by removing duplicated subtrees [15]. 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:

(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, we can only select *one* of these sets to be handled by *remove_duplicate*. Furthermore, for this selected set $\mathcal{S}$, even if $\mathcal{S}$ 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 cannot be applied repeatedly for removing the remaining duplicated subtrees in $\mathcal{S}$, nor the remaining sets of subtrees duplicated elsewhere.

This paper addresses the above two issues. The rest of the paper is structured as follows. Section 2 reviews some previous and related work on the classification-tree method. Section 3 describes in detail our new restructuring algorithm. Section 4 compares our algorithm with *remove_duplicate*, and provides guidelines on the appropriate restructuring algorithm for a given type of classification trees. Finally, Section 5 concludes the paper.

2 Previous work on the classification-tree method

2.1 *Original work*

The classification-tree method [12, 13] was developed by Grochtmann et al. based on the category-partition method [10, 11] as the fundamental framework. It helps testers to generate test cases from specifications via the construction of classification trees. Basically, a classification tree organizes the classifications and classes into a tree structure. The following describes the major steps of the method:

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

(2) Identify the associated classes for every classification.

(3) Organize all the identified classifications and classes into a classification tree.

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

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

(6) 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.

Suppose the classifications and classes for *ticket* are identified as in Table 1. As 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 $3\times4\times3\times3\times4\times3\times3 = 3\,888$ test cases will be produced.[1] However, some of these test cases are invalid because of the coexistence 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".

In order to reduce the number of invalid test cases, a classification tree is constructed. For instance, a classification tree for *ticket*, denoted by $\mathcal{T}_{ticket}$, 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 "Rank of Staff" and "Type of Ticket" 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, "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 Figure 1, for example, "Domestic" is the parent class of "Mileage",

[1] The number of possible ways to select no more than one class from any classification with n number of associated classes is $n+1$ because none of these n classes may be selected.

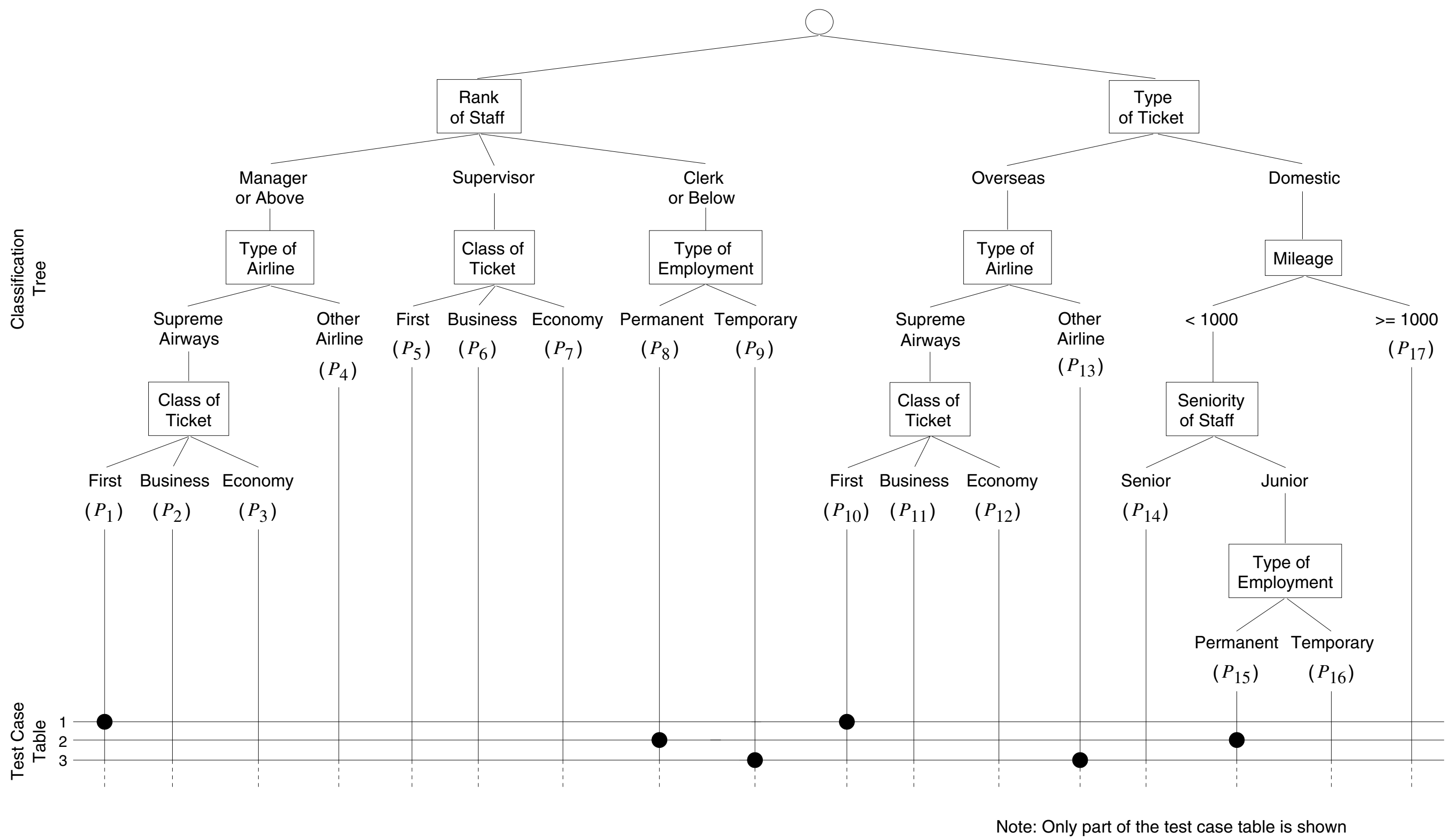

Figure 1: *Classification tree for ticket and part of the test case table*

Classifications	Associated Classes
Seniority of Staff	Senior, Junior
Rank of Staff	Manager or Above, Supervisor$^{(1)}$, Clerk or Below$^{(1)\,(2)\,(3)}$
Type of Employment	Permanent, Temporary
Type of Airline	Supreme Airways, Other Airline$^{(2)\,(4)\,(5)}$
Class of Ticket	First, Business, Economy
Type of Ticket	Overseas$^{(6)}$, Domestic$^{(1)}$
Mileage	< 1000, ≥ 1000

(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”

Table 1: Possible classifications and classes for *ticket*

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 Figure 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:

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

(2) Construct a 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 1 of the test case table in Figure 1 represents the potential test case {Manager or Above, Supreme Airways, First, Overseas}.

We use P_i to denote a path in $\mathcal{T}_{ticket}$. 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 Figure 1. The complete table produces a total of 72 potential test cases. Compared to the 3 888 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 3 816 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 $\mathcal{T}_{ticket}$. ■

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 [12], an identification system for automatic mail sorting machines [12], an integrated ship management system [12] and a simplified part of an adaptive cruise control system [16]. 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:

(1) Using more formal specifications, or

(2) Systematically capturing constraints among various classifications.

For approach (1), a well-known study is one performed by Singh et al. [16]. 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 (2). 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 [12] 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 [14]. 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 $\mathcal{T}_{ticket}$ of Figure 1, 57 are illegitimate. Only 15 potential test cases are legitimate and therefore useful for testing.

In [15], 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 $\mathcal{T}$, let $N[\mathcal{T},p]$ and $N[\mathcal{T},l]$ be the number of potential test cases and legitimate test cases, respectively. Chen and Poon defined an *effectiveness metric* $E[\mathcal{T}]$ for $\mathcal{T}$ as follows:

$$E[\mathcal{T}] = \frac{N[\mathcal{T},l]}{N[\mathcal{T},p]} \qquad (1)$$

For example, since $N[\mathcal{T}_{ticket},p] = 72$ and $N[\mathcal{T}_{ticket},l] = 15$, $E[\mathcal{T}_{ticket}]$ is found to be $\frac{15}{72} = 0.21$. Obviously, $N[\mathcal{T},l]$ 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, $N[\mathcal{T},p]$ can be derived directly from $\mathcal{T}$ using the formulae presented in [15].

Apparently, a small value of $E[\mathcal{T}]$ 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 $E[\mathcal{T}]$.

In [15], Chen and Poon proposed that a major cause for a poor $E[\mathcal{T}]$ 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 $\mathcal{T}_{ticket}$ of Figure 1, the subtree S[Type of Airline] is duplicated because it occurs in the top-level subtrees S[Rank of Staff] and S[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 $\{(P_1 \text{ or } P_2 \text{ or } P_3) \text{ and } P_{13}\}$ or the set of paths $\{P_4 \text{ and } (P_{10} \text{ or } P_{11} \text{ or } P_{12})\}$. Similarly, since "Type of Employment" is related to both the top-level classifications "Rank of Staff" and "Type of Ticket", the subtree S[Type of Employment] is also duplicated in both the top-level subtrees. These two duplications result in 57 illegitimate test cases, thereby reducing $E[\mathcal{T}_{ticket}]$ to a very small value.

From this observation, Chen and Poon developed a tree restructuring algorithm *remove_duplicate* to improve on the value of $E[\mathcal{T}]$ for classification trees with duplicated subtrees under different top-level classifications [15]. 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 [1] is increased from 0.17 to 0.40. Readers may refer to [15] 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 [15]. 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 [15]:

(*a*) The algorithm assumes that classification trees do not have duplicated subtrees under the *same* top-level classification. Consider, for example, the classification tree $\mathcal{T}_{ticket}$ in Figure 1. The subtree S[Class of Ticket] appears twice under the top-level classification "Rank of Staff". The algorithm cannot remove such duplications.

(*b*) 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 $\mathcal{S}$ handled by *remove_duplicate*, even if $\mathcal{S}$ 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 Figure 2. Suppose that, in this classification tree,

- τ_1, τ_2 and τ_3 denote the top-level subtrees $S[A]$, $S[B]$ and $S[G]$, respectively,
- $S_{\tau_1}[C]$ and $S_{\tau_2}[C]$ denote the subtrees within τ_1 and τ_2, respectively, with the classification C as their roots, and
- $S_{\tau_1}[F]$, $S_{\tau_2}[F]$ and $S_{\tau_3}[F]$ denote the subtrees within τ_1, τ_2 and τ_3 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*) $\{S_{\tau_1}[C],\ S_{\tau_2}[C]\}$

(*ii*) $\{S_{\tau_1}[F],\ S_{\tau_2}[F],\ S_{\tau_3}[F]\}$

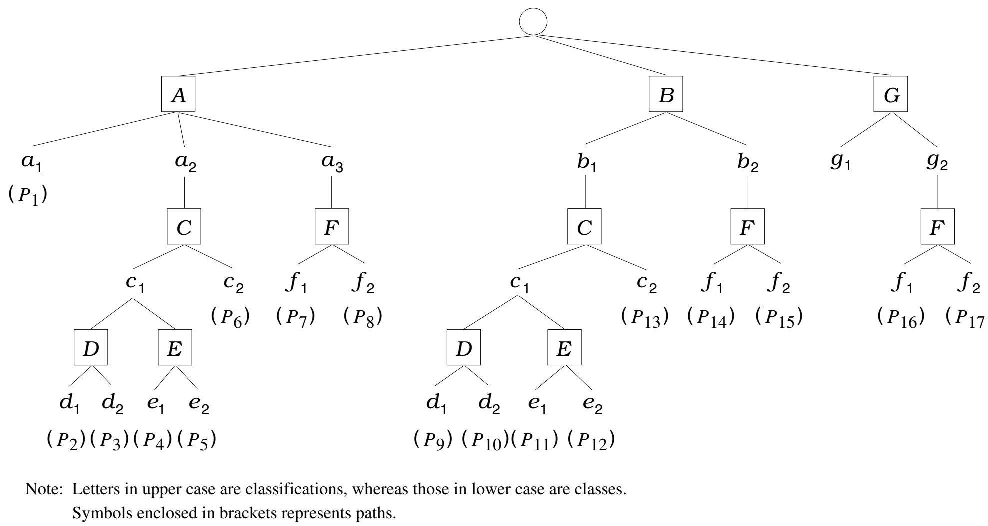

Figure 2: *A classification tree with two sets of duplicated subtrees*

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. Obviously, the algorithm can be automated without difficulty.

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

Let

- τ_i, $i = 1,\ 2,\ \ldots,\ w$ be the w top-level subtrees of a classification tree $\mathcal{T}$, where $w \geq 2$,
- $S_{\tau_i}[X]$ be a subtree of τ_i such that the root of $S_{\tau_i}[X]$ is the classification X,
- τ_i' be the top-level subtree formed from τ_i after pruning from it all the subtrees $S_{\tau_i}[X]$, and
- $N(\tau_i)$ and $N(\tau_i')$ be the total number of combinations of classes for τ_i and τ_i', respectively.

Suppose two or more of the top-level subtrees contain duplicated subtrees. Without loss of generality, let these top-level subtrees be $\tau_1, \tau_2, \ldots, \tau_n$, where $2 \leq n \leq w$, and let the duplicated subtrees be $S_{\tau_1}[X], S_{\tau_2}[X], \ldots, S_{\tau_n}[X]$.[2] Suppose, further, that the situation is as follows:

- For every duplicated classification Y in $\mathcal{T}$, all the subtrees with classification Y as their roots are identical.

[2]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, such as $\mathcal{T}_{ticket}$ in Figure 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.

- Let classification U be a descendent of classification V in $\mathcal{T}$ (that is, U is placed under some class of V in $\mathcal{T}$). For any class v of V, if v can be combined with some class of U to form part of a legitimate test case, U will appear under this class v.

Then:

> Select a top-level subtree τ_k (where $1 \le k \le n$) such that, if we prune all the $S_{\tau_1}[X]$, $S_{\tau_2}[X]$, …, $S_{\tau_{k-1}}[X]$, $S_{\tau_{k+1}}[X]$, …, $S_{\tau_n}[X]$ from τ_1, τ_2, …, τ_{k-1}, τ_{k+1}, …, τ_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 τ_1, τ_2, …, τ_{k-1}, τ_{k+1}, …, τ_n by τ'_1, τ'_2, …, τ'_{k-1}, τ'_{k+1}, …, τ'_n, respectively, but leave the selected top-level subtree τ_k unchanged. In case there are two or more distinct τ_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 τ_j and τ_k. Note, however, that $S_{\tau_k}[X]$ is allowed to occur more than once *within* a top-level subtree τ_k.

In the above algorithm, $N(\tau'_j)$ and $N(\tau_k)$ can be derived using the formulae from [15]. Suppose $\mathcal{T}'$ denotes the classification tree after restructuring. According to the formulae for the computation of $N[\mathcal{T}', p]$ in [15], the smaller the value of Q, the smaller will be the value of $N[\mathcal{T}', p]$. 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 not exceed 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. ∎

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 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 Figure 1, the potential test case corresponding to row 2 is formed by selecting the feasible net that contains the paths $\{P_8,\ P_{15}\}$.

Let τ_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 feasible paths $\mathcal{P}$ in $\mathcal{F}$ such that $\mathcal{P}$ is within τ_i. Suppose, for instance, τ_1 denotes the top-level subtree in Figure 2 with classification A as its root. Then, $\mathcal{F}|_{\tau_1} = \{P_2,\ P_4\}$ is a feasible subnet within τ_1.

Now, suppose a classification tree has two or more top-level classifications denoted by τ_1, τ_2, …, τ_w. Suppose further that:

(*a*) τ_i and τ_j (where $i \neq j$ and $1 \le i,\ j \le w$) denote two distinct top-level subtrees containing duplicated subtrees $S_{\tau_i}[X]$ and $S_{\tau_j}[X]$, respectively, and (*b*) τ_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 $S_{\tau_k}[X]$.

The feasible subnets within τ_i can be classified as follows:

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

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

Let us illustrate this concept using the classification tree in Figure 2. Again, let τ_1 denote the top-level subtree $S[A]$. Then, $\{P_2,\ P_4\}$ is a feasible subnet in $\boldsymbol{F}(\tau_1,\ C)$, and $\{P_1\}$ is a feasible subnet in $\boldsymbol{F}(\tau_1,\ \neg C)$.

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 $\mathcal{T}$ has been restructured using remove_identical to form $\mathcal{T}'$. Any legitimate test case that can be constructed from $\mathcal{T}$ can also be constructed from $\mathcal{T}'$.

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 $\mathcal{T}$ has w top-level subtrees denoted by τ_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 $S_{\tau_1}[X]$, $S_{\tau_2}[X]$, ..., $S_{\tau_n}[X]$, respectively,

(*c*) for any $1 \leq i \leq n$, τ_i' denotes the top-level subtree formed by pruning all the subtrees of the form $S_{\tau_i}[X]$ from τ_i, and

(*d*) after the application of *remove_identical*, all the duplicated subtrees in (*b*) are removed, except for the subtree(s) $S_{\tau_k}[X]$ in one and only one top-level subtree τ_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 τ_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 $\boldsymbol{F}(\tau_i,\ X)$, $i = 1,\ 2,\ \ldots,\ n$, and one feasible subnet from every τ_i, $i = n+1,\ n+2,\ \ldots,\ w$.

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

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

(*ii*) The potential test case is formed by selecting feasible subnets from a mixture of $\boldsymbol{F}(\tau_i,\ X)$ and $\boldsymbol{F}(\tau_j,\ \neg X)$ for $i,\ j = 1,\ 2,\ \ldots,\ n$ (where $i \neq j$), and one feasible subnet from every τ_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 $\mathcal{F}|_{\tau_j}$ selected from $\boldsymbol{F}(\tau_j,\ \neg X)$. By definition, any path in $\mathcal{F}|_{\tau_j}$ 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 subset from $\boldsymbol{F}(\tau_i,\ X)$, 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 $\mathcal{T}$ will remain in $\mathcal{T}'$ after restructuring.

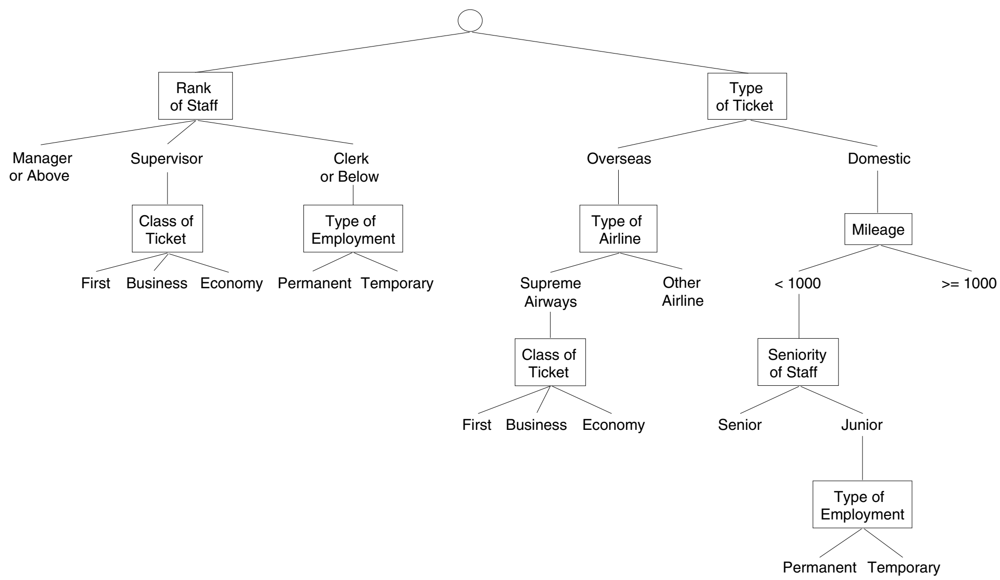

Figure 3: *The resultant classification tree $\mathcal{T}_{ticket}$ after pruning S_{τ_1}[Type of Airline]*

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

Such a test case, whether legitimate or otherwise, will remain unchanged after restructuring because:

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

Because of this, any legitimate test case that can be constructed from $\mathcal{T}$ can also be constructed from $\mathcal{T}'$. ■

Let us use Example 2 to illustrate the application of the restructuring algorithm *remove_identical* and to show the improvement in $E[\mathcal{T}]$.

Example 2 (Restructuring of Classification Trees)
Consider the classification tree $\mathcal{T}_{ticket}$ in Figure 1. Let τ_1 and τ_2 denote the top-level subtrees S[Rank of Staff] and S[Type of Ticket], respectively.

Consider the duplicated subtrees S_{τ_1}[Type of Airline] and S_{τ_2}[Type of Airline]. There are two alternate ways of restructuring $\mathcal{T}_{ticket}$ using the algorithm *remove_identical*:

(a) Prune S_{τ_1}[Type of Airline] from τ_1, or

(b) Prune S_{τ_2}[Type of Airline] from τ_2.

Figures 3 and 4 depict the two classification trees after the above ways of restructuring, respectively. Let τ'_1 be the result of pruning S_{τ_1}[Type of Airline] from τ_1, and τ'_2 be that of pruning S_{τ_2}[Type of Airline] from τ_2. Using the formulae presented in [15], $N(\tau'_1) \times N(\tau_2) = 48$ for Figure 3 and $N(\tau_1) \times N(\tau'_2) = 45$ 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_{τ_1}[Type of Employment] and $S_{\tau'_2}$[Type of Employment]. The restructuring algorithm *remove_identical*

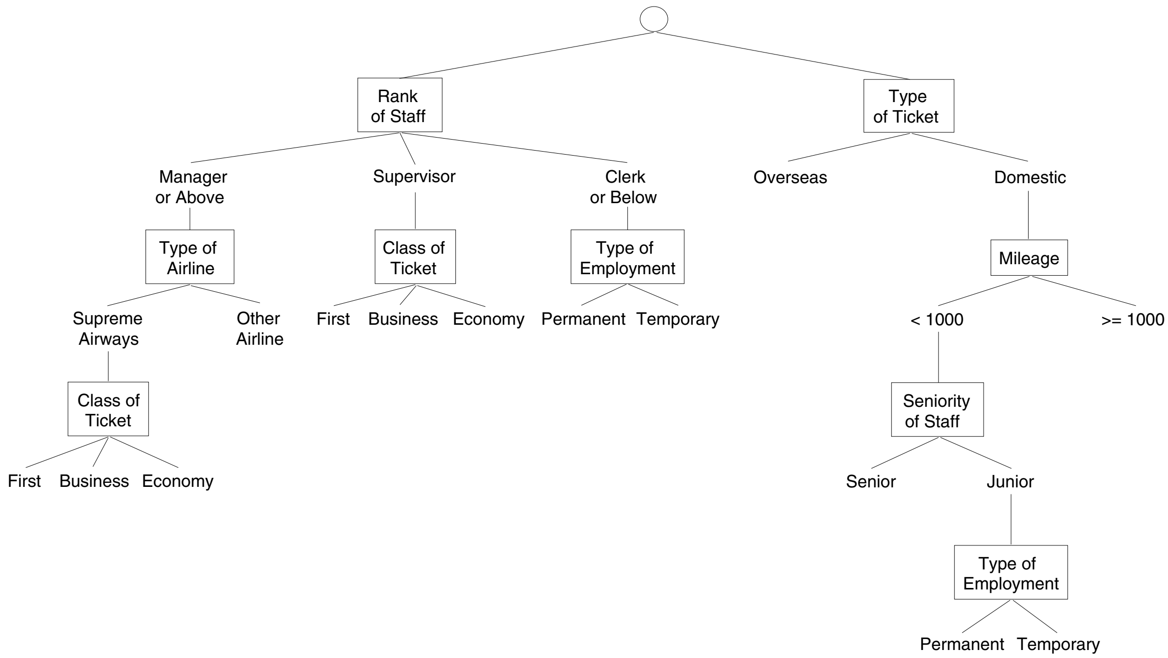

Figure 4: *The resultant classification tree $\mathcal{T}_{ticket}$ after pruning $S_{\tau_2}[\textit{Type of Airline}]$*

should therefore be applied again to remove duplications. The resultant classification tree $\mathcal{T}'_{ticket}$ after the second application is depicted in Figure 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 $\mathcal{T}_{ticket}$ before restructuring can still be constructed from $\mathcal{T}'_{ticket}$. Hence, $N[\mathcal{T}'_{ticket}, l] = N[\mathcal{T}_{ticket}, l] = 15$. On the other hand, $N[\mathcal{T}'_{ticket}, p]$ is calculated to be 36 using the formulae in [15]. Thus, $E[\mathcal{T}'_{ticket}] = \frac{15}{36} = 0.42$. Compared to $E[\mathcal{T}_{ticket}] = 0.21$, the improvement is 100% and therefore quite significant. ■

4 A comparison between the two restructuring algorithms

Both the new restructuring algorithm *remove_identical* and Chen and Poon's previous algorithm *remove_duplicate* have their own merits. We would like to compare them in this section.

(a) With regard to the classification tree in Figure 2, for instance, the new algorithm *remove_identical* is preferred to *remove_duplicate*. It is because both sets of duplicated subtrees $\mathcal{S}_1 = \{S_{\tau_1}[C],\ S_{\tau_2}[C]\}$ and $\mathcal{S}_2 = \{S_{\tau_1}[F],\ S_{\tau_2}[F],\ S_{\tau_3}[F]\}$ [3] can be handled by *remove_identical*, whereas only one of these sets can be handled by *remove_duplicate*. Furthermore, in case of $\mathcal{S}_2$, *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 $\mathcal{T}_{ticket}$ in Figure 1.

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

[3]Note that τ_1, τ_2 and τ_3 denote the top-level subtrees with classifications A, B and G as their roots, respectively.

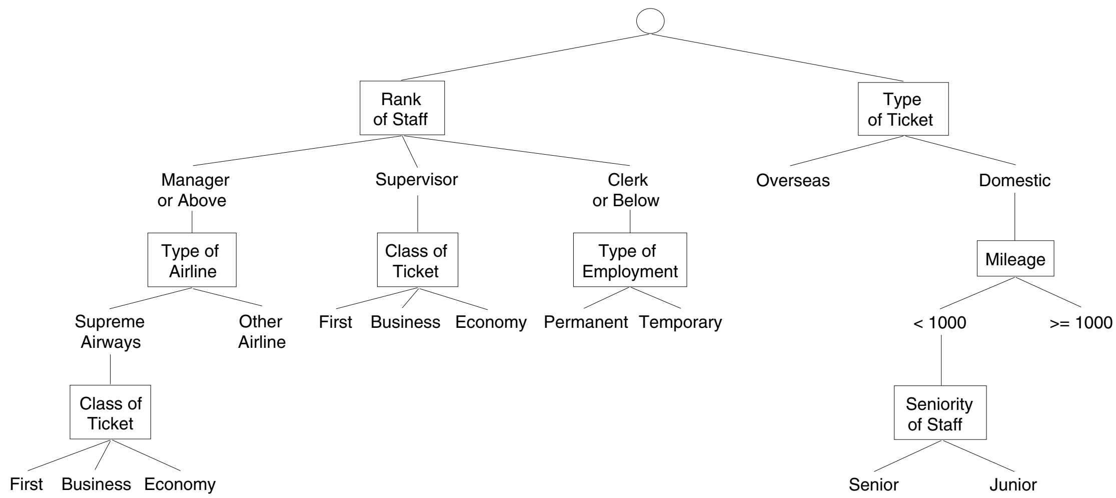

Figure 5: *The final classification tree $\mathcal{T}'_{ticket}$ after restructuring*

(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 $E[\mathcal{T}]$ 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 $\mathcal{S}$,
- (ii) There are only *two* duplicated subtrees in $\mathcal{S}$,
- (iii) The two duplicated subtrees occur across different top-level classifications, and
- (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 $E[\mathcal{T}]$. 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 $\mathcal{T}_1$ in Figure 6. Using the formulae in [15], we find that the total number of potential test cases $N[\mathcal{T}_1, p]$ is 12. We note that the structure of $\mathcal{T}_1$ satisfies all the conditions (i) to (iv) in (d) above. It can be restructured by both the algorithms *remove_identical* and *remove_duplicate*.

Figure 7 depicts the classification tree $\mathcal{T}_1'$ after applying the algorithm *remove_identical*, and Figure 8 depicts the tree $\mathcal{T}_1''$ after applying *remove_duplicate*. According to Proposition 2, $N[\mathcal{T}_1, l] = N[\mathcal{T}_1', l]$. According to [15], $N[\mathcal{T}_1, l] = N[\mathcal{T}_1'', l]$. Hence, $N[\mathcal{T}_1', l] = N[\mathcal{T}_1'', l]$. Using the formulae in [15], we find that $N[\mathcal{T}_1', p] = 8$ and $N[\mathcal{T}_1'', p] = 4$. Since $N[\mathcal{T}_1', l] = N[\mathcal{T}_1'', l]$ and $N[\mathcal{T}_1', p] > N[\mathcal{T}_1'', p]$, we have $E[\mathcal{T}_1'] < E[\mathcal{T}_1'']$ by Equation 1 quoted from [15]. 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 $\mathcal{T}$, suppose

(i) *$\mathcal{T}$ contains only one set of duplicated subtrees denoted by $\mathcal{S}$,*

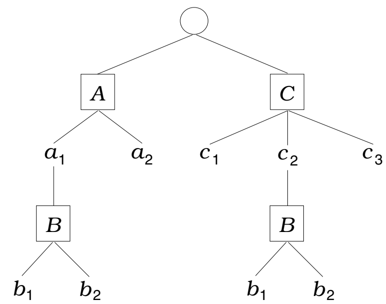

Figure 6: *A classification tree $\mathcal{T}_1$ with one set of duplicated subtrees*

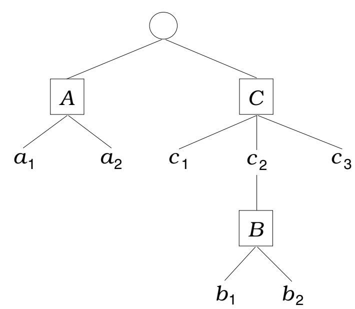

Figure 7: *A classification tree $\mathcal{T}_1'$ after restructuring $\mathcal{T}_1$ using remove_identical*

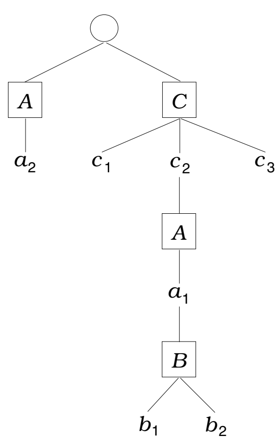

Figure 8: *A classification tree $\mathcal{T}_1''$ after restructuring $\mathcal{T}_1$ using remove_duplicate*

(ii) *$\mathcal{S}$ contains only two duplicated subtrees,*

(iii) *The two duplicated subtrees occur across different top-level classifications, and*

(iv) *For every duplicated classification X in $\mathcal{T}$, all the subtrees with X as their roots are identical.*

Let

(a) *$\mathcal{T}'$ denote the tree after applying the restructuring algorithm $remove\_identical$, and*

(b) *$\mathcal{T}''$ denote the tree after applying the restructuring algorithm $remove\_duplicate$ instead.*

Then $N[\mathcal{T}'', p] \leq N[\mathcal{T}', p]$.

Proof
We shall follow the notation used in the restructuring algorithm $remove\_identical$. Without loss of generality, suppose that

- $\mathcal{T}$ has w top-level subtrees denoted by τ_i, $i = 1,\ 2,\ \ldots,\ w$, where $w \geq 2$,
- $\mathcal{T}$ contains only one set of duplicated subtrees $\{S_{\tau_m}[X],\ S_{\tau_n}[X]\}$, where $1 \leq m,\ n \leq w$ and $m \neq n$,
- τ_m contains *one and only one* $S_{\tau_m}[X]$,
- τ_n contains *one and only one* $S_{\tau_n}[X]$, and
- for any duplicated classification Y in $\mathcal{T}$, 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 $\mathcal{T}$.
For $remove\_identical$, suppose

- $S_{\tau_m}[X]$ is to be pruned from τ_m, and
- τ_m and τ_n are restructured into τ'_m and τ'_n, respectively.

Obviously, $\tau'_n = \tau_n$ and hence $N(\tau'_n) = N(\tau_n)$.
Consider $remove\_duplicate$. Let

- the parent of X in τ_m is the class z in the classification Z, and
- τ_m and τ_n are restructured into τ''_m and τ''_n, respectively.

Furthermore, suppose the sequence of restructuring of the classification tree $\mathcal{T}$ is as follows:

(1) The subtree Λ_1 is formed by pruning $S_{\tau_m}[X]$ from $S_{\tau_m}[Z]$.

(2) The duplicated subtree $S_{\tau_n}[X]$ is replaced by the subtree $\Lambda_2 = Z — z — S_{\tau_m}[X]$.

(3) $S_{\tau_m}[Z]$ is replaced by a null tree, or Λ_1, or a modified Λ_1 with z deleted from it, depending on its initial structure before pruning $S_{\tau_m}[X]$ from it in step (1).

It can be seen from step (2) that $N(\Lambda_2) = N(S_{\tau_m}[X]) = N(S_{\tau_n}[X])$. Hence, $N(\tau''_n) = N(\tau_n)$. Since $N(\tau'_n) = N(\tau_n)$, we must have $N(\tau''_n) = N(\tau'_n)$.

Now, consider the three cases for the replacement of $S_{\tau_m}[Z]$ in step (3):

Case A: $S_{\tau_m}[Z]$ is replaced by a null tree. This is equivalent to the pruning of $S_{\tau_m}[Z]$ from τ_m.

In this case, $S_{\tau_m}[X]$ must be the unique subtree of z, and z must be the unique child class of Z. Hence, the reduction of $N(\tau_m)$ will be the same no matter whether we prune $S_{\tau_m}[Z]$ as recommended by the restructuring algorithm *remove_duplicate*, or prune $S_{\tau_m}[X]$ as recommended by *remove_identical*. Thus, we must have $N(\tau''_m) = N(\tau'_m)$.

Case B: $S_{\tau_m}[Z]$ is replaced by Λ_1

In this case, X cannot be the unique child classification of z. It is obvious from the formation of Λ_1 in step (1) that $\tau''_m = \tau'_m$. Hence, we must have $N(\tau''_m) = N(\tau'_m)$.

Case C: $S_{\tau_m}[Z]$ is replaced by a modified Λ_1 with z deleted from it. This is equivalent to the pruning of $S_{\tau_m}[z]$ from τ_m.

In this case, $S_{\tau_m}[X]$ must be the unique subtree of z, and z cannot be the unique child class of Z. Since $S_{\tau_m}[X]$ is a subtree of $S_{\tau_m}[z]$, the reduction of $N(\tau_m)$ by pruning $S_{\tau_m}[z]$ as recommended by *remove_duplicate* must be greater than that by pruning $S_{\tau_m}[X]$ as recommended by *remove_identical*. Hence, we must have $N(\tau''_m) < N(\tau'_m)$.

Since $N(\tau''_m) \leq N(\tau'_m)$, $N(\tau''_n) = N(\tau'_n)$, and all the top-level subtrees τ_i (where $1 \leq i \leq w$, $i \neq m$ and $i \neq n$) will remain intact after the application of *remove_identical* or *remove_duplicate* to $\mathcal{T}$, we must have $N[\mathcal{T}'', p] \leq N[\mathcal{T}', p]$. ■

Proposition 3 indicates that *remove_duplicate* rather than *remove_identical* should be used as the restructuring mechanism when conditions (i) to (iv) in Section 4 are fulfilled. More specifically, the following guideline should be used for restructuring:

Apply *remove_duplicate* if

(a) $\mathcal{T}$ contains only *one* set of duplicated subtrees denoted by $\mathcal{S}$,

(b) There are only *two* duplicated subtrees in $\mathcal{S}$, and

(c) The two duplicated subtrees occur across different top-level classifications rather than within the same top-level classification.

Otherwise, apply *remove_identical*.

5 Conclusion

Testing plays an important role in verifying the correctness of software. As the quality of testing largely depends on the comprehensiveness of test cases [5, 6, 2, 3, 11], it is essential to have a systematic method for constructing test cases from specifications. The classification-tree method developed by Grochtmann et al. [12] 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 [14]. They 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 further 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. From these observations, they (i) defined an effectiveness metric for classification trees, and (ii) 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 restructure a given classification tree.

6 Acknowledgment

This work is supported in part by the Research Grants Council of Hong Kong (Project Nos. CityU 1118/99E and HKU 7029/01E).

References

[1] CHEN, T.Y., POON, P.L., and TSE, T.H.: 'A new restructuring algorithm for the classification-tree method'. Proceedings of the 9th International Workshop on Software Technology and Engineering Practice, STEP '99, August–September 1999, Pittsburgh, Pennsylvania, pp. 105–114

[2] FERGUSON, R., and KOREL, B.: 'The chaining approach for software test data generation', *ACM Trans. Softw. Eng. Methodol.*, 1996, **5**, (1), pp. 63–86

[3] KOREL, B.: 'Automated test data generation for programs with procedures'. Proceedings of the International Symposium on Software Testing and Analysis, ISSTA '96, January 1996, San Diego, California, pp. 209–215

[4] SANDERS, J., and CURRAN, E.: 'Software quality: a framework for success in software development and support' (Addison Wesley, Wokingham, UK, 1994)

[5] BACHE, R., and MÜLLERBURG, M.: 'Measures of testability as a basis for quality assurance', *Softw. Eng. J.*, 1990, **5**, (3), pp. 86–92

[6] CHUSHO, T.: 'Test data selection and quality estimation based on the concept of essential branches for path testing', *IEEE Trans. Softw. Eng.*, 1987, **SE-13**, (5), pp. 509–517

[7] STOCKS, P., and CARRINGTON, D.: 'A framework for specification-based testing', *IEEE Trans. Softw. Eng.*, 1996, **22**, (11), pp. 777–793

[8] CHEN, H.Y., TSE, T.H., CHAN, F.T., and CHEN, T.Y.: 'In black and white: an integrated approach to class-level testing of object-oriented programs', *ACM Trans. Softw. Eng. Methodol.*, 1998, **7**, (3), pp. 250–295

[9] CHEN, H.Y., TSE, T.H., and CHEN, T.Y.: 'TACCLE: a methodology for object-oriented software testing at the class and cluster levels', *ACM Trans. Softw. Eng. Methodol.*, 2001, **10**, (1), pp. 56–109

[10] BALCER, M.J., HASLING, W.M., and OSTRAND, T.J.: 'Automatic generation of test scripts from formal test specifications'. Proceedings of the 3rd ACM Annual Symposium on Software Testing, Analysis, and Verification, TAV '89, December 1989, Key West, Florida, pp. 210–218

[11] OSTRAND, T.J., and BALCER, M.J.: 'The category-partition method for specifying and generating functional tests', *Comm. ACM*, 1988, **31**, (6), pp. 676–686

[12] GROCHTMANN, M., and GRIMM, K.: 'Classification trees for partition testing', *Softw. Testing, Verification and Reliability*, 1993, **3**, (2), pp. 63–82

[13] GROCHTMANN, M., WEGENER, J., and GRIMM, K.: 'Test case design using classification trees and the classification-tree editor CTE'. Proceedings of the 8th International Software Quality Week, QW '95, May 1995, San Francisco, California

[14] CHEN, T.Y., and POON, P.L.: ‘Construction of classification trees via the classification-hierarchy table’, *Inf. Softw. Technol.*, 1997, **39**, (13), pp. 889–896

[15] CHEN, T.Y., and POON, P.L.: ‘On the effectiveness of classification trees for test case construction’, *Inf. Softw. Technol.*, 1998, **40**, (13), pp. 765–775

[16] SINGH, H., CONRAD, M., and SADEGHIPOUR, S.: ‘Test case design based on Z and the classification-tree method’. Proceedings of the 1st IEEE International Conference on Formal Engineering Methods, ICFEM ’97, November 1997, Hiroshima, Japan, pp. 81–90

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 a permanent or 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

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 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 kg, 30 kg and 20 kg, respectively.
- (ii) In the case of "Domestic" discounted tickets:
  - If the mileage is less than 1000 and the employee is a senior staff member, then the maximum weight of baggage is 15 kg.
  - 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.
  - 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.
  - 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.