A "Strength of Decision Tree Equivalence"-Taxonomy and Its Impact on Test Suite Reduction

Hermann Felbinger, Ingo Pill, Franz Wotawa

A "Strength of Decision Tree Equivalence"-Taxonomy and Its Impact on Test Suite Reduction

Hermann Felbinger, Ingo Pill, and Franz Wotawa

Institute for Software Technology
Graz University of Technology, Austria,
{felbinger,ipill,wotawa}@ist.tugraz.at

Abstract. Being able to reduce test suites without having to execute them for assessing the effects on their fault detection capabilities is quite appealing. In this direction, we proposed recently to characterize test suites via inferred decision trees and use these for comparisons in a reduction process. The equivalence relation underlying the comparisons plays obviously a significant role for the effectiveness achieved and efficiency experienced. In this paper, we explore five such relations that take different aspects into account and investigate their impact on test suite reduction, their effectiveness in fault detection, and computation time. We report corresponding results, and show as well as prove that the equivalence relations build a taxonomy.

Keywords: Test Suite Reduction, Decision Tree Equivalence

1 Introduction

Today, our software tends to be improved and extended almost constantly during its life cycle. Correspondingly, also the test suites we use for their validation tend to grow with new product features, the isolation of faults to be avoided in the future, and with the advent of new concepts for generating tests effective at unveiling specific software issues. The impact of software testing on the overall development costs, however, demands keeping test suites as small as possible while preserving their fault detection capabilities. Consequently, we need effective test suite reduction approaches in order to manage resources and costs related to a test suite's execution, validation, and management.

Even when focusing on predefined faults (like for mutation testing [9]) such that we knew exactly which faults some test case t can identify, finding a minimum sized test suite able to identify a maximum of faults, is an instance of the set cover problem that is one of Karp's 21 NP-complete problems [19]. Still, drawing on effective heuristics, researchers faced the challenge and proposed various strategies to tackle the problem, e.g., [4, 12, 13, 15, 22, 24]. Known strategies rely, e.g., on existing links between requirements and test cases, on analyzing

execution traces that can cover others, or on preserving coverage and mutation scores as indicators for a test suite's effectiveness.

An attractive feature of the approach introduced by Felbinger et al. in [10] is that we do not need to execute the program under test for assessing the fault detection capabilities when removing a test case. The underlying idea was that every test suite T should at least partially capture the behavior of the program under test in a sufficient way. The strategy then is to use machine learning for model extraction, in order to derive representative characterizations from T and a reduced test suite T'. We proposed in [10] the following reduction process: Initially, we learn a characterizing decision tree from T, and when successively trying to remove test cases $t \in T$, we infer for each potential removal another decision tree from the updated test suite T'. If the decision tree for T' is *equivalent* to the initial one, we assume that the fault detection capabilities were not affected, and proceed with trying to remove further test cases. Otherwise, we go back one step and re-add t. The reduction terminates after a preconfigured number of unsuccessful, random tries to remove a further test case. With avoiding to execute T, we still could achieve reductions from 60 to 99% in our evaluation. Our approach for decision tree learning is limited to test cases t represented as some vector $t = \langle x_1, .., x_k, out \rangle$ of k input values and an expected output value *out*. The inputs are either numeric of an infinite domain, numeric of a finite domain, or discrete strings or numbers. The output type has to be of a finite domain, whose values then build the labels of the decision trees' leaf nodes.

Since such a test suite reduction depends on an equivalence relation for decision trees, the following questions arise immediately: Which methods are there for determining equivalence? Are there more than structural and misclassification equivalence as discussed and used in [10] (coined syntactic and semantic equivalence there), and is there a relation between them? What is their impact on the efficiency and effectiveness of the reduction process?

Imagining variants, one has to take the characteristics of the derived trees into account. According to [17], optimizing a decision tree to a minimal number of nodes which would allow us to compare minimal or canonical ones, is in NP. Thus, the algorithm used to infer the decision trees in [10] is based on a statistical measure (the information gain of variables) and does not stringently build optimal decision trees. Consequently, trees inferred from different test suites might appear different in respect of their strict structure. Exploring flexibility in this respect, we consider five variants for checking some trees' equivalence. In particular, we consider in Section 4 structural ($\equiv$), spine ($=^s$), decision ($=^d$), table ($=^t$), and misclassification ($=^m$) equivalence aiming to cover and explore various decision tree aspects. We show and prove that these variants build a taxonomy as shown in Figure 1 in respect of their strength. We report in Section 5 on our corresponding experiments, considering computation time and the achieved reductions as well as the impact on fault detection capabilities. In Section 6 we conclude on our findings and line out future work.

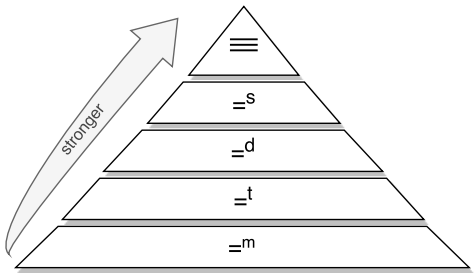

Fig. 1. Taxonomy of equivalence relation in respect of their strength.

2 Related Work

Safavian and Landgrebe provide a survey of decision tree classifiers in [26]. They address the design, search strategies, issues like missing values and robustness, and potential problems of decision trees in their survey. In [21] Moret provides a common framework of definitions and notations for decision trees. In [7] Dattatreya and Kanal introduce the usage of decision trees in pattern recognition. In this context they define pattern recognition as "the assignment of a physical object or an event to one of the prespecified categories". They consolidate the major methodologies for decision tree design, bring out those methodologies' commonalities, provide insight into multistage classification, explode the myth that decision trees are always simple to design and use, mention areas of applications of decision trees, and aid a decision tree designer to select an appropriate technique for the particular problem of interest.
Cockett introduces in [6] different notions of decision tree equivalence. These notions are structural, decision, and transposition equivalence that are similar to some of the notions we use in this work, which are structural, spine, and decision equivalence, but Cockett uses the notion of coalgebras to describe decision trees and the equivalence relations. In [28] Zantema presents a simple efficient algorithm to establish whether two decision trees are equivalent or not. This algorithm is an axiomatization for decision equivalence as we use it in this work. The complexity of this algorithm is bounded by the product of the number of nodes n and m of both decision trees ($O(n * m)$). The algorithm only processes decision trees representing discrete valued variables as decision nodes. In our work we also cover numeric inputs, which are handled by binary splits. The authors in [5] present an algorithm that reduces a decision tree by replacing the decision tree with a smaller equivalent decision tree. To find an irreducible tree using the reduction algorithm they also use decision and transposition equivalence. In [29] the authors address the question, whether for a given decision tree, a decision tree decision equivalent to the given one can be found, for which no decision equivalent decision tree of smaller size exists. Breslow and Aha provide an overview over methods how to simplify a decision tree in [2].

The underlying idea that a model inferred from a test suite can be used to indicate the fault detection effectiveness of the test suite was initially published in [11]. In [11] Felbinger et al. show that a linear correlation between model inference based test suite quality assessment without executing the program under test might depend on the structural properties, the types of inputs, and the number of discrete outputs of the program under test. Some initial results of test suite reduction without executing the program under test are provided in [10]. The promising results in [10], where reductions of 60-99% were possible, while still keeping coverage and mutation score almost the same, led to this work, where we used the same reduction algorithm. In [10] structural and misclassification equivalence were used to obtain the results. Briand et al. [3] describe a test suite refinement approach that relies on the black box testing technique Category-Partition [23] and machine learning. They use categories and choices to define the functional properties of a program under test, where categories are associated with choices. E.g. a category representing an inequality relation has two choices of an inequality relation that are either greater than or less than. Based on these categories they transform test cases into abstract test cases. These abstractions are tuples of choices and an expected output value or an equivalence class of expected output values. Like in our work, they use the C4.5 algorithm [25] to learn a decision tree in [3]. But in contrast to our work, where we learn a decision tree from the raw values in a test suite, they learn decision trees from the abstractions obtained by category-partitioning.

Since test suite reduction has been of interest for decades, there is a tremendous amount of further related work. We refer the interested reader to [1] and [27] for detailed overviews.

3 Preliminaries

In our work, we infer a decision tree D from a test suite T via the well-known algorithm C4.5 [25]. Such a decision tree is a directed tree $D = (V, E)$ having nodes V and directed edges E connecting nodes. V can be split into decision nodes and leaf nodes, where a decision node has outgoing edges and represents a decision (i.e., a relational equation) like $x > 0$ (see Fig. 2) for some numeric input x, or x *equals* $\langle$*discrete value*$\rangle$ for discrete inputs. A leaf node is a terminal one and offers a discrete classification. An edge (v, v') is a pair of nodes $(v, v' \in V)$, where v is parent of v'. For simplicity, we assume a function ρ: $DT \rightarrow V$ that returns the root node of a decision tree, with the universe of decision trees DT under consideration as input domain. Further we assume a function λ: $V \rightarrow J \cup C$ that returns the content of a node, with the union of the set of decisions J and the set of classifications C as range. The decision trees in this work are binary such that each decision node has exactly two outgoing edges. The answer of a decision, e.g., whether we have $x > 0$, is represented by an edge label that can be accessed via a function γ: $E \rightarrow \{T, F\}$. In our decision trees, paths are sequences containing nodes and connecting edges, starting from the root node,

following down the tree, and ending at a leaf node. We define a path Π in a decision tree as follows:

Definition 1 (Path). *A path Π of length $|\Pi| = l$ in a decision tree D is a sequence of nodes $v_0...v_{l-1}$ such that there is an edge from v_i to v_{i+1} for $0 \leq i < l-1$, starting with $v_0 = \rho(D)$ and ending at a leaf node v_{l-1}.*

With C4.5, decision trees are constructed top down, where decision nodes get selected using a statistical property called *information gain* that measures how well a decision separates the $t \in T$ according to their expected outcome [20]. A test case t is classified in a decision tree by following the decision nodes from the root node, down the tree to some leaf node, according to the values in t. Not necessarily all input variables appear in a decision tree, but numeric variables can occur also multiple times in different decision nodes, even in the same path. We define equivalence for decision trees as follows:

Definition 2 (Equivalence Relation). *Decision Tree Equivalence is a reflexive, symmetrical, and transitive binary relation R between two decision trees D_1 and D_2 from the universe of decision trees DT, such that:*
reflexivity: $\forall D \in DT : D\, R\, D$
symmetry: $\forall D_1, D_2 \in DT : D_1\, R\, D_2 \rightarrow D_2\, R\, D_1$
trans.: $\forall D_1, D_2, D_3 \in DT : D_1\, R\, D_2 \wedge D_2\, R\, D_3 \rightarrow D_1\, R\, D_3$

In our work, we consider the equivalence of decision trees when reducing test suites. When trying to remove test cases from a test suite T without effecting changes in the decision tree, the achieved reduction is an indicator of the reduction process' effectivity:

Definition 3 (Reduction). *Given a test suite T and a reduced test suite $T' \subseteq T$, the achieved reduction is defined via the difference in their sizes:*

$$reduction = \frac{|T| - |T'|}{|T|} \tag{1}$$

When we infer a decision tree, we derive a hypothesis h regarding an approximation of a function f that we can use to predict f's outcome for future input values. Strategies for estimating the accuracy of such a hypothesis include k-folds cross validation [16], or assessment with additional input and output values [20]. In principle, for evaluating a hypothesis h, we can use the function $error(h, S)$ as given in Equation 2 in order to obtain a result in the range 0..1:

$$error(h, S) = \frac{1}{|S|} \sum_{t \in S} \delta(f(t), h(t)) \tag{2}$$

Equation 2 requires three parts: First, some set S that should be different to T (from which the hypothesis was learned) containing vectors t of input values and an expected output. Second, the target function $f : I^k \rightarrow O$, where I is the type of the k inputs and O represents the set of all possible outputs. Third, a function δ that detects deviating outcomes of f and h–returning 1 if $f(t) \neq h(t)$ for some $t \in S$ and 0 otherwise.

4 Equivalence Taxa

For our investigation, we considered five decision tree equivalence relations, ranging from structural equivalence to misclassification equivalence. Before showing at the end of this section that they form a taxonomy in respect of their strength, let us formally introduce them for the decision trees $D_1 = (V_1, E_1)$ and $D_2 = (V_2, E_2)$ first.

Structural Equivalence ($\equiv$): Two decision trees D_1 and D_2 are structurally equivalent, if and only if each node $v_1 \in V_1$ has a corresponding node $v_2 \in V_2$ and each edge $e_1 \in E_1$ has a corresponding edge $e_2 \in E_2$ connecting an equivalent pair of nodes. Structural equivalence can be represented using a function EQUAL: $V \times V \rightarrow \{True, False\}$, which we define recursively as follows: For two decision trees D_1, D_2, and nodes $v_1 \in V_1$, $v_2 \in V_2$, EQUAL returns *True*, if and only if:

1. $\lambda(v_1) = \lambda(v_2)$
2. $\forall(v_1, v_i) \in E_1,\ \exists(v_2, v_j) \in E_2,\ 0 \leq i, j < 2|$
   $\gamma(v_1, v_i) = \gamma(v_2, v_j) \wedge \text{EQUAL}(v_i, v_j)$ *(and vice versa)*

Using this function, we define structural equivalence of two decision trees as follows:

Definition 4 (Structural equivalence). *Two given decision trees D_1, D_2 are structurally equivalent if and only if the function* $\text{EQUAL}(\rho(D_1), \rho(D_2))$ *returns True.*

EQUAL terminates if it detects different node contents or different edge labels, or if all nodes have been visited.

Example 1 (Structural equivalence). Figure 2 shows two structurally equivalent decision trees where decision nodes, leaf nodes, and edges are equivalent and on the same position in both decision trees.

Spine Equivalence ($=^s$): A decision tree consists of a set of spines *SP*. A spine $(\Pi, c) \in SP$ is described by a path Π to a leaf node v, such that $c = \lambda(v)$. Spine equivalence requires bag equivalence to hold, which is defined as:

Definition 5 (Bag equivalence). *Two paths Π_1 and Π_2 are equivalent as bags, if except for the ordering they contain nodes with precisely the same content and with equivalently labelled outgoing edges, such that for all $v_1 \in \Pi_1$ there exists an equivalent node $v_2 \in \Pi_2$ and vice versa, where $\lambda(v_1) = \lambda(v_2)$ and $\gamma(v_1, v_i) = \gamma(v_2, v_j)$.*

From the definitions of a path and bag equivalence, we define spine equivalence as:

Definition 6 (Spine equivalence). *Two decision trees D_1 and D_2 are spine equivalent if for the respective sets of spines SP_1 and SP_2, for every spine $(\Pi_1, c_1) \in SP_1$ there exists a spine $(\Pi_2, c_2) \in SP_2$ and vice versa, such that Π_1 and Π_2 are bag equivalent and $c_1 = c_2$.*

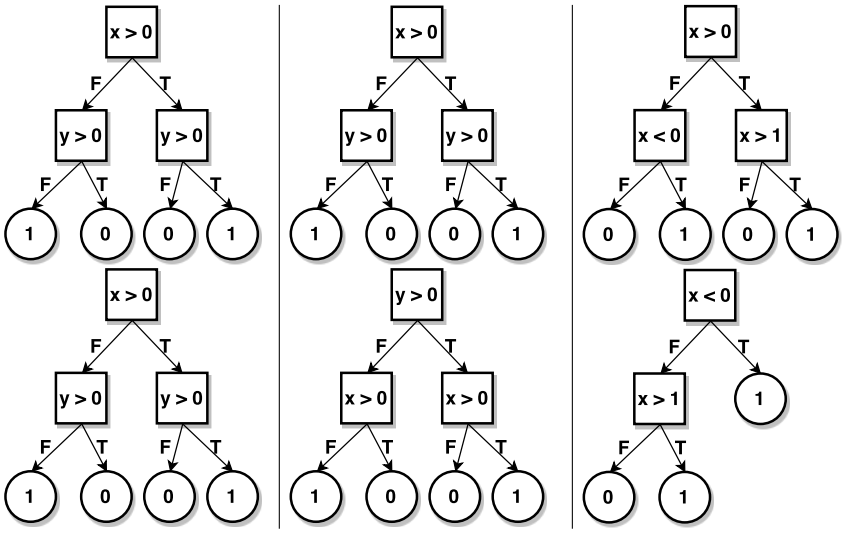

Fig. 2. Structurally (left), spine- (middle), and decision-equivalent (right) trees.

Example 2 (Spine equivalence). Figure 2 shows two spine equivalent decision trees where the order of decision nodes in the paths differ, but the spines are equivalent. However, these decision trees are not structurally equivalent.

Decision Equivalence ($=^d$): A constraint built from a spine's path is a conjunction of equivalence relations that contain a decision node's content and its outgoing edge's label for all decision nodes in the path. Satisfying a constraint classifies the inputs to that spine's c. In a decision tree, there may be multiple spines for some c. For decision equivalence, we thus build a summarizing constraint for each c as a disjunction of the corresponding conjunctions of the individual spines for c. E.g., from the top right decision tree in Figure 2, a constraint ψ of paths from spines with $c = 1$ is $(x > 0 = F \wedge x < 0 = T) \vee (x > 0 = T \wedge x > 1 = T)$. More formally, we define decision equivalence as:

Definition 7 (Decision equivalence). *Two decision trees D_1 and D_2 are decision equivalent, if for all leaf nodes $v_1 \in V_1$ an equivalent leaf node $v_2 \in V_2$ exists, and for each constraint ψ_1 of D_1 there exists a constraint ψ_2 in D_2 where the following equation holds:*

$$\psi_1 \textit{ equals } \psi_2 \tag{3}$$

Equation 3 is true, if no valuation exists for which ψ_1 is satisfiable and ψ_2 is unsatisfiable, and vice versa.

Example 3 (Decision equivalence). Figure 2 shows two decision equivalent decision trees that do not contain the same decision nodes and are therefore not spine equivalent.

Table Equivalence ($=^t$): A decision tree D classifies all test cases $t \in T$ according to the input values in t to a leaf node, as introduced in Section 3. A test case t is misclassified, if $\lambda(v)$ for the leaf node v to which t was classified and the value *out* of t differ. Otherwise t is classified correctly. These principle is also used in hypothesis evaluation as introduced in Section 3, where the function $h(t)$ returns the content of the leaf node to which t was classified, but unlike for hypothesis evaluation, here the outcome of the target function $f(t)$ is the value of *out* that is already included in t. We create a set M of pairs $(h(t), out)$ that contains for each $t \in T$ the content of the leaf node to which t was classified and the value *out* of t. Note that we have $|M| = |T|$. Two sets M_1 and M_2 are equivalent, if for each pair $(h(t)_1, out_1) \in M_1$ there is a pair $(h(t)_2, out_2) \in M_2$ such that $h(t)_1 = h(t)_2$ and $out_1 = out_2$, and vice versa. Consequently, we define table equivalence as:

Definition 8 (Table Equivalence). *Two decision trees D_1 and D_2 are table equivalent, when classifying a test suite T yields two equivalent sets M_1 for D_1 and M_2 for D_2.*

Example 4 (Table equivalence). Figure 3 shows two decision trees that are table equivalent if T does not contain a test case $t = \langle 1, 1, 1 \rangle$, because then $h(t) \neq out$ only for the lower decision tree. These decision trees are not decision equivalent.

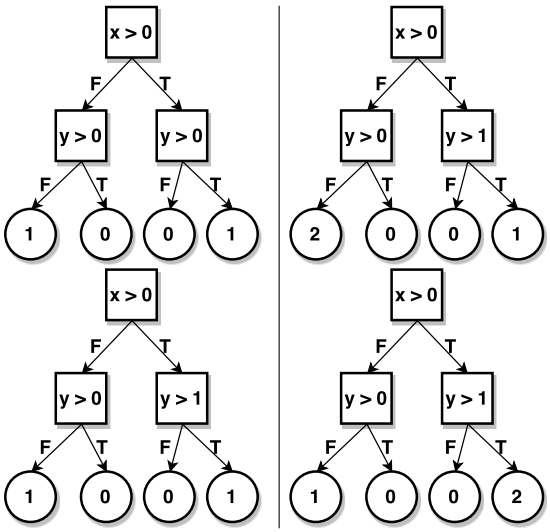

Fig. 3. Table (left) and misclassification-equivalent (right) trees.

Misclassification Equivalence ($=^m$): Two decision trees D_1 and D_2 are equivalent regarding their misclassification rate $error(D, T)$, if the following two conditions hold:

1. $error(D_2, T) = error(D_1, T)$.
2. For all distinct contents in the leaf nodes $C_1 \subset V_1$ an equivalent classification exists in the leaf nodes $C_2 \subset V_2$ and vice versa.

Definition 9 (Misclassification equivalence). *A decision tree D_2 is misclassification equivalent to a reference decision tree D_1, when classifying T, if the following equation holds:*

$$\begin{gathered} error(D_2, T) = error(D_1, T) \wedge \\ \forall v_1 \in C_1, \exists v_2 \in C_2 | v_1 = v_2 \textit{ (and vice versa)} \end{gathered} \tag{4}$$

Example 5 (Misclassification equivalence). Figure 3 shows two decision trees where the same classifications exist in both decision trees as visualized by the leaf nodes. If T contains two test cases $t_1 = \langle 0, 0, 2 \rangle$ and $t_2 = \langle 1, 2, 2 \rangle$, the decision trees are misclassification equivalent, but not table equivalent.

Theorem 1. *The five defined methods to determine equivalence of decision trees can be presented in a subset order, where for a decision tree D inferred from a test suite T, subsets $DT_{\equiv} \subset DT$, $DT_{=^s} \subset DT$, $DT_{=^d} \subset DT$, $DT_{=^t} \subset DT$, and $DT_{=^m} \subset DT$ from the universe of decision trees DT exist, which contain decision trees that were inferred from a test suite $T' \subseteq T$, and are equivalent to D. These subsets are ordered as*
$DT_{\equiv} \subseteq DT_{=^s} \subseteq DT_{=^d} \subseteq DT_{=^t} \subseteq DT_{=^m}$, for subsets
$DT_{\equiv} \subset DT$ representing structural equivalent decision trees,
$DT_{=^s} \subset DT$ representing spine equivalent decision trees,
$DT_{=^d} \subset DT$ representing decision equivalent decision trees,
$DT_{=^t} \subset DT$ representing table equivalent decision trees, and
$DT_{=^m} \subset DT$ representing misclassification equivalent decision trees.

Proof. (sketch) Structural equivalence implies that all paths are equivalent. If all paths are equivalent, spine equivalence is ensured. If paths in two decision trees only have different orders of nodes, the decision trees are spine equivalent, but not structurally equivalent. Spine equivalence implies that all nodes contain the same content. Building constraints from the paths in spines ensures that the constraints are equivalent, because they contain the same contents of nodes and the same outgoing edges of the decision nodes. If a node is missing or redundant in a path of two decision equivalent decision trees, this contradicts spine equivalence. Decision equivalence implies that each possible input valuation leads to an equivalent classification or misclassification. Equivalent classifications for all possible input values ensure table equivalence, because table equivalence depends only on the classification of a test suite T, which contains only a subset of all possible input values. If a pair of decision trees is table equivalent, but test cases are missing for boundary values of the decisions, different decision nodes in a path lead to equivalent classifications for a test suite T, but not for each possible input valuation. This fact contradicts decision equivalence. Table equivalence implies that two decision trees provide equivalent classifications for a test

suite T, independently of whether a test case was correctly classified or misclassified. If all test cases in T are equally classified or misclassified, misclassification equivalence is given. A misclassification equivalent pair of decision trees where classifying a test suite T yields the same misclassification rate, but different test cases from T are misclassified, violates table equivalence. ■

As stated in Theorem 1, structural equivalence is the strongest method to determine equivalence of two decision trees, meaning that even if the four other equivalence check methods evaluate to true, structural equivalence can be false. Decision equivalence is the costliest method due to the NP-completeness of determining inequality of two constraints. Misclassification equivalence is the weakest method to determine equivalence of two decision trees, because it neither considers the structure of the decision tree nor the relation of inputs to outputs.

5 Experimental Evaluation

We used three different Java programs for our proof-of-concept experiments, generated combinatorial test suites using the tool ACTS[1], and evaluated the reduced test suites' fault detection effectiveness via their mutation score. For generating mutants, we used the Major mutation framework [18].

5.1 Results

The three examples are Triangle, TCAS, and UTF8, as introduced in [10]. For test suite reduction, we implemented the *REDUCE* algorithm from [10] in Java and instantiated the *equals* method in *REDUCE* at line 12 by all 5 equivalence methods introduced in Section 4. The input values for *iterations* and *retries* of *REDUCE* were set to 2 and $\frac{|T|}{10}$ respectively, since the results in [10] show that these values allow high reductions. To infer decision trees from a test suite, we used the Java library Weka [14] and its implementation J48 of the algorithm C4.5. In the configuration options of Weka, we disabled pruning and set the minimum number of leaf nodes to 1. The expected outcome for a test case was derived with the original program. We obtained test suites of size 343 (Triangle), 1840 (UTF8), and 11021 (TCAS), and generated 35 (Triangle) and 147 (UTF8) mutants. For the TCAS example, we used the 41 existing mutants[2]. In order to determine decision equivalence, we applied the SMT-solver Z3 [8] that provides a Java-API. For calculating the misclassification rate, we used the decision tree evaluation method integrated in the Weka library. Since the *REDUCE* algorithm selects potentially redundant test cases randomly, we executed the algorithm for each example 10 times per equivalence method and plot the execution time and the resulting reduction for each execution. All experiments ran on a MacBook Pro with an Intel Core i5 2.7GHz CPU, 16GB RAM, an SSD, and OS X 10.11.6.

[1] http://csrc.nist.gov/groups/SNS/acts
[2] http://sir.unl.edu/portal/bios/tcas.php

The resulting reductions and the runtime to obtain these reductions for the Triangle example are shown in Figure 4. The results in Figure 4 show that decision equivalence is multiple times slower than other equivalence methods. Structural equivalence is fastest, misclassification and table equivalence allow the highest reductions. Reductions of structural equivalence are lowest. The results in Figure 5 for the UTF8 example show that structural and spine equivalence are fastest, but the reductions are around 30% lower than for the other equivalence methods. Also for the UTF8 example decision equivalence was slowest. For the TCAS example the results in Figure 6 show that all reductions only vary in a range of around 10%. Also for TCAS, structural and spine equivalence are fastest and decision equivalence is slowest on average. The highest reductions were obtained by table and misclassification equivalence.

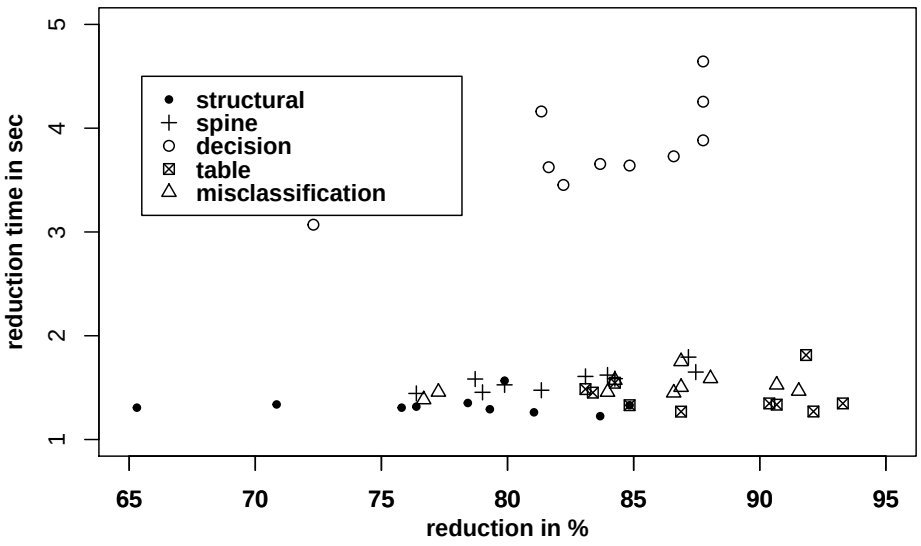

Fig. 4. Triangle results.

5.2 Discussion

Our results suggest that structural equivalence, whose complexity is linear in the number of nodes in a decision tree, is the fastest and decision equivalence is the slowest equivalence method. Deciding decision equivalence is an NP-complete problem and each pair of equivalent constraints in two decision trees gives the worst case. When using misclassification equivalence, which allows the highest reductions, the time to reduce T was slightly higher than for structural equivalence. For evaluating a potential loss of the test suite's fault detection effectiveness, we derived the mutation score for all reduced test suites as reported in

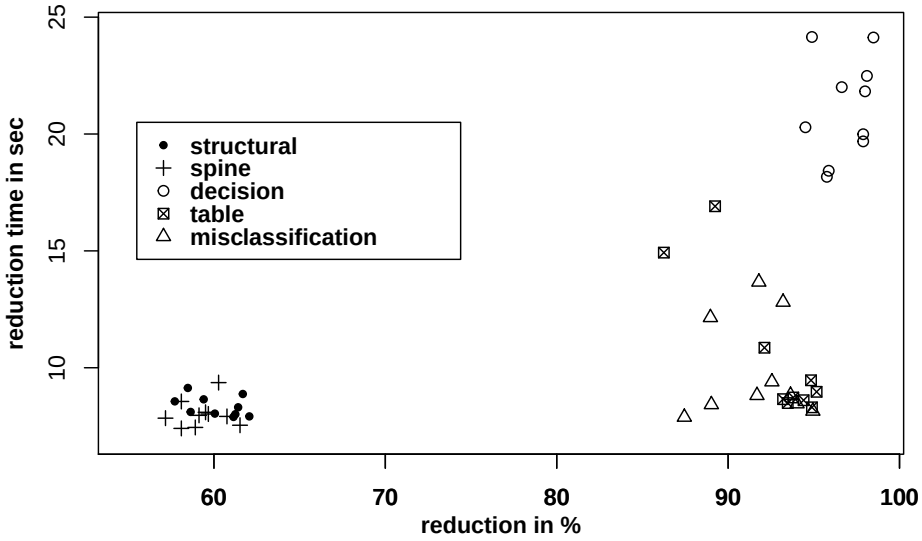

Fig. 5. UTF8 results.

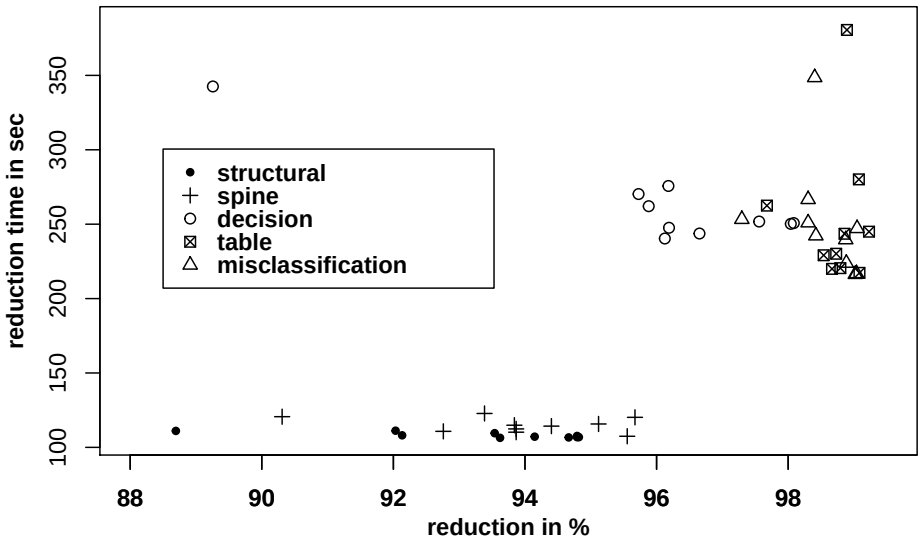

Fig. 6. TCAS results.

Figure 7. The mutation score of the initial test suites was 1 for each example. For each example in Figure 7, the equivalence methods are ordered according to their strength from left to right, starting on the left with the strongest one. The results show that for the strongest equivalence method there was almost no decline of mutation score, but for weaker methods the mutation score decreased. In particular for the UTF8 example, the median mutation score dropped to values in the range 0.6 to 0.7 for decision, table, and misclassification equivalence. These weak mutation scores origin in the fact that the initial test suite contained test cases with unknown values, which were approximated automatically while inferring a tree by the C4.5 algorithm. These approximations increased potential uncertainties of the tree to predict future outputs for additional input values. The dots in the plots for Triangle, UTF8, and TCAS represent outliers from the obtained results.

Using structural or spine equivalence provided similar reductions at similar costs. Although decision equivalence allowed high reductions, the computation time was highest from all equivalence methods. Table and misclassification equivalence provided the highest reduction results for our examples, consuming more time than structural and spine equivalence (but in most cases less time than decision equivalence). The mutation score results suggest the highest loss of fault detection effectiveness to occur when using table or misclassification equivalence. Therefore, if the execution time of the tests in the finally reduced test suite is low, structural equivalence should be chosen. If keeping the fault detection capabilities as high as possible for a reduced test suite, also structural equivalence should be chosen. In all other cases the results suggest that misclassification equivalence is an educated choice. Promising results of an empirical evaluation of structural and misclassification equivalence were provided in [10]. With our results, we clarify that the runtime of the reduction approach depends on three parts. First, the runtime depends on the size of the test suite and the domain sizes of the inputs. The latter affects the run-time spent for the algorithm C4.5, since we have to learn a decision tree for each potentially removable test case. Second, as we surmised, the runtime depends on the complexity of the equivalence relation used. Last, but not least, we saw that the runtime increases also with the achieved reduction.

6 Conclusion

In this paper, we introduce a "strength of decision tree equivalence"-taxonomy of five different equivalence relations. Decision tree equivalence is a crucial part of a recently introduced test suite reduction approach that does not require to execute the program under test. We came up with five different methods to determine this equivalence and provide a theorem and a corresponding proof that these methods form a taxonomy in respect of their strength. As a proof of concept, our experiments show that the equivalence method indeed has a high impact on the effectiveness and efficiency when reducing a test suite. The results yield structural and spine equivalence as the methods with the lowest costs, but

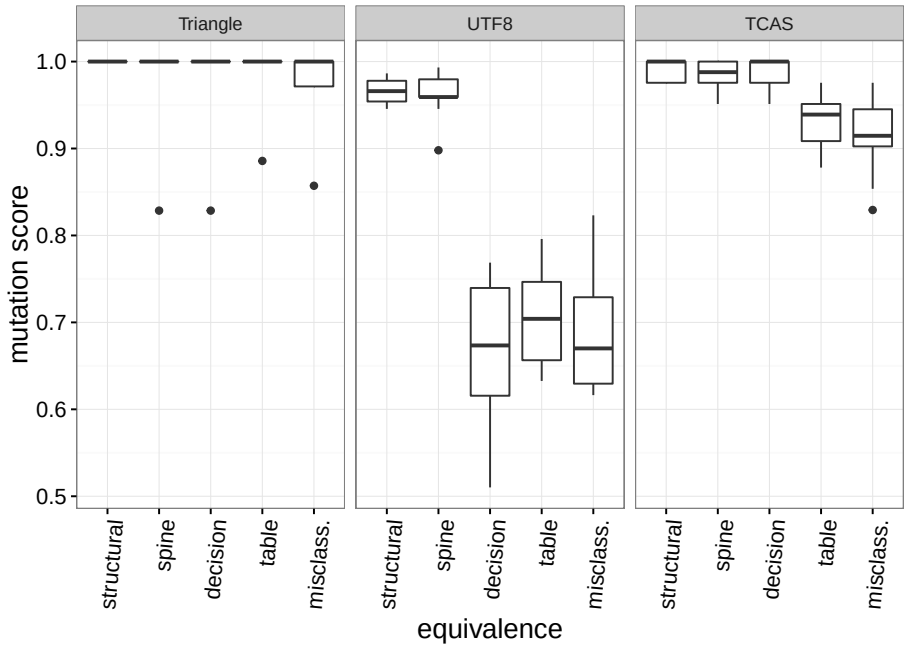

Fig. 7. Mutation score of reduced test suites.

also with the smallest reduction. Decision equivalence is the costliest in respect of computation time, but achieves high reductions. When determining equivalence with table and misclassification equivalence, the reductions are very high, but suffer from the highest decrease in fault detection effectiveness.

Underpinning the reduction approach itself and the selection of the most appropriate equivalence relation will require an evaluation with additional, realistic scenarios. If some T does not contain redundancies, no reduction is possible. For detecting that T does not contain redundancies structural equivalence should be chosen, because it is the least time consuming relation to determine. Since the structure (control flow, data flow, lines of code, etc.) of the program under test affects the reduction, with more examples possibly a classification can be created such that we could derive from the program structure in combination with background information on how T was generated which equivalence method would be best suited.

For our current experiments, we used first order mutants for evaluating the effectiveness in fault detection, but towards applicability of the reduction approach in practice, an examination with higher order mutants shall be part of future work. In future work we will extend also our empirical evaluation, considering more examples from application domains like automotive control software. Here an open research question is also how such a program's structure affects the test suite reduction approach of [10] in general.

Acknowledgment

We thank the ECSEL Joint Undertaking (supported by the EU Horizon 2020 programme and the ECSEL member states) for funding this work under grant agreement 662192 (3Ccar). This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and the ECSEL member states.

References

1. Biswas, S., Mall, R., Satpathy, M., Sukumaran, S.: Regression test selection techniques: A survey. Informatica 35(3), 289–321 (2011)
2. Breslow, L.A., Aha, D.W.: Simplifying decision trees: A survey. The Knowledge Engineering Review 12(1), 1–40 (Jan 1997)
3. Briand, L.C., Labiche, Y., Bawar, Z.: Using machine learning to refine black-box test specifications and test suites. In: 8th Int. Conf. on Quality Software. pp. 135–144 (2008)
4. Chen, T.Y., Lau, M.F.: Dividing strategies for the optimization of a test suite. Information Proc. Letters 60(3), 135–141 (1996)
5. Cockett, J.R.B., Herrera, J.A.: Decision tree reduction. Journal of the ACM 37(4), 815–842 (Oct 1990)
6. Cockett, J.: Discrete decision theory: manipulations. Theoretical Computer Science 54(2), 215–236 (1987)
7. Dattatreya, G., Kanal, L.: Progress in Pattern Recognition 2, chap. Decision Trees in Pattern Recognition, pp. 189–240. Elsevier Science Publishers B.V. (1985)
8. De Moura, L., Bjørner, N.: Z3: An Efficient SMT Solver. In: 14th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340 (2008)
9. DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on test data selection: Help for the practicing programmer. Computer 11(4), 34–41 (Apr 1978)
10. Felbinger, H., Wotawa, F., Nica, M.: Test-suite reduction does not necessarily require executing the program under test. In: Int. Conf. on Software Quality, Reliability and Security Companion. pp. 23–30 (2016)
11. Felbinger, H., Wotawa, F., Nica, M.: Empirical study of correlation between mutation score and model inference based test suite adequacy assessment. In: 11th Int. Workshop on Automation of Software Test. pp. 43–49 (2016)
12. Fraser, G., Wotawa, F.: Redundancy based test-suite reduction. In: Int. Conf. on Fund. Approaches to Software Eng. pp. 291–305 (2007)
13. Gotlieb, A., Marijan, D.: Flower: Optimal test suite reduction as a network maximum flow. In: 2014 Int. Symposium on Software Testing and Analysis (ISSTA). pp. 171–180 (2014)
14. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explorations Newsletter 11(1), 10–18 (Nov 2009)
15. Harrold, M.J., Gupta, R., Soffa, M.L.: A methodology for controlling the size of a test suite. ACM Transactions on Software Engineering and Methodology (TOSEM) 2(3), 270–285 (Jul 1993)
16. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer Series in Statistics, Springer New York Inc. (2009)

17. Hyafil, L., Rivest, R.: Constructing optimal binary decision trees is NP-complete. Information Proc. Letters 5(1), 15–17 (1976)
18. Just, R.: The major mutation framework: Efficient and scalable mutation analysis for java. In: Proceedings of the 2014 International Symposium on Software Testing and Analysis. pp. 433–436. ACM (2014)
19. Karp, R.M.: Reducibility among Combinatorial Problems, pp. 85–103. Springer US (1972)
20. Mitchell, T.M.: Machine learning, vol. 8. McGraw-Hill Boston, MA (1997)
21. Moret, B.M.: Decision trees and diagrams. ACM Computing Surveys (CSUR) 14(4), 593–623 (1982)
22. Offutt, A.J., Pan, J., Voas, J.M.: Procedures for reducing the size of coverage-based test sets. In: Proceedings of the 12th International Conference on Testing Computer Software. pp. 111–123. ACM (1995)
23. Ostrand, T.J., Balcer, M.J.: The category-partition method for specifying and generating fuctional tests. Communications of the ACM 31(6), 676–686 (Jun 1988)
24. Polo Usaola, M., Reales Mateo, P., Pérez Lamancha, B.: Reduction of test suites using mutation. In: 15th Int. Conf. on Fundamental Approaches to Software Engineering. pp. 425–438 (2012)
25. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
26. Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier methodology. IEEE Transactions on Systems, Man, and Cybernetics 21(3), 660–674 (May 1991)
27. Yoo, S., Harman, M.: Regression testing minimization, selection and prioritization: A survey. Software Testing, Verifification And Reliability 22(2), 67–120 (2012)
28. Zantema, H.: Decision trees: Equivalence and propositional operations. In: 10th Netherlands/Belgium Conf. on AI (NAIC). pp. 157–166 (1998)
29. Zantema, H., Bodlaender, H.L.: Finding small equivalent decision trees is hard. International Journal of Foundations of computer science 11(2), 343–354 (2000)