Managing Conflicts when Using Combination Strategies to Test Software

Mats Grindal
Humanities and Informatics
University of Skövde
Sweden
mats.grindal@his.se

Jeff Offutt
Info. and Software Engineering
George Mason University
Fairfax, VA 22030, USA
offutt@ise.gmu.edu

Jonas Mellin
Humanities and Informatics
University of Skövde
Sweden
jonas.mellin@his.se

Abstract

Testers often represent systems under test in input parameter models. These contain parameters with associated values. Combinations of parameter values, with one value for each parameter, are potential test cases. In most models, some values of two or more parameters cannot be combined. Testers must then detect and avoid or remove these conflicts.

This paper proposes two new methods for automatically handling such conflicts and compares these with two existing methods, based on the sizes of the final conflict-free test suites. A test suite reduction method, usable with three of the four investigated methods is also included in the study, resulting in seven studied conflict handling methods.

In the experiment, the number and types of conflicts, as well as the size of the input parameter model and the coverage criterion used, are varied. All in all, 3854 test suites with a total of 929,158 test cases were generated.

Two methods stand out as tractable and complementary. The best method with respect to test suite size is to avoid selection of test cases with conflicts. However, this method cannot always be used. The second best method, removing conflicts from the final test suite, is completely general.

1. Introduction

The input space of a test problem can often be described by the parameters of the "program under test" or "system under test." However, the number of parameters and the possible values of each parameter usually results in too many combinations to be useful. *Combination strategies* is a class of test case selection methods that use combinatorial strategies to select test suites that have tractable size.

A large number of combination strategies have been proposed and are surveyed in our previous paper [6]. A common property of all combination strategies is that they all require input parameter models. An *input parameter model* (IPM) is an abstract description of the input space and possibly the state of the system under test.

An IPM consists of a set of modeling parameters with a set of values for each parameter. The *IPM parameters* are based on functions of the actual parameters of the system under test (SUT). The state of the SUT may also be used in these functions. A special, and common case, is when the SUT parameters are mapped one-to-one onto the IPM parameters. An *IPM parameter value* represents a non-empty subset of values of the input space and possibly the state of the system under test.

The category partition method [12] is one structured way to create input parameter models. Based on the IPM, combination strategies generate test cases by selecting one value from each IPM parameter and combining these into abstract SUT inputs. A set of test cases is selected and together they form a test suite. The test cases are selected based on some notion of coverage and the objective of the combination strategy is to select the test cases in the test suite such that 100% coverage is achieved. 1-*wise* (also known as each-used) coverage is the simplest coverage criterion. Each-used coverage requires that every value of every parameter is included in at least one test case in the test suite. 2-*wise* (also known as pair-wise) coverage requires that every possible pair of values of any two parameters are included in some test case. Note that the same test case can often cover more than one unique pair of values. A natural extension of 2-wise coverage is t-*wise* coverage, which requires every possible combination of interesting values of t parameters be included in some test case in the test suite.

The most thorough coverage criterion, N-*wise* coverage, requires a test suite to contain every possible combination of the parameter values in the IPM. The resulting test suite is often too large to be practical.

The creation of IPMs sometimes results in conflicts between IPM parameter values of two or more IPM parameters. A *conflict* is an impossible combination of two or more IPM parameter values.

As an example of IPMs with conflicts, consider

testing the function `boolean find (element, vector)`, which returns true if the element is found in the vector. Figure 1 shows a partial IPM for testing the find function.

The IPM parameters and their associated values are given unique names. This logical representation is used to make the implementations of the combination strategies and conflict handling methods completely general. This also allows conflicts to be described as pairs of IPM parameter values.

This example uses letters to denote IPM parameters and integers to represent the values of each parameter. For instance, "$A1$" in the example shown in Figure 1 is short for an "empty vector."

In the IPM in Figure 1, the size of the vector and the number of times an element occurs in the vector are modeled as two different IPM parameters. This model has three impossible sub-combinations. For instance $(A2, B3)$ represents a situation where an element should occur more than once in a vector with only one element.

A (Size of vector) = {**1**:0, **2**:1, **3**:>1}
B (Element occurs) = {**1**:No, **2**:Once, **3**:More than once}
C (Sorting) = {**1**:Unsorted, **2**:Sorted, **3**:All identical}
Impossible sub-comb.: (A1, B2), (A1, B3), (A2, B3)

Figure 1. IPM for testing a find function.

The purpose of conflict handling is to give the tester the power to exclude impossible sub-combinations, which is not the same as eliminating negative test cases, such as the empty vector in the find example.

In their basic forms, combination strategies generate test suites that satisfy the desired coverage without using any semantic information. Hence, impossible test cases may be selected as part of the test suite. How to handle these conflicts has not previously been fully investigated. Important questions include which conflict handling methods work for which combination strategies and how the size of the test suite is affected by the conflict handling method.

An important **principle** of conflict handling is that the coverage criterion must be preserved. That is, if a test suite satisfies 1-wise coverage with conflicts, it must still satisfy 1-wise coverage after the conflicts are removed. All conflict handling methods in this study satisfy this principle. An important **goal** is minimize the growth in the number of tests in the test suite.

Two methods to handle this situation have previously been proposed. In the "sub-models" method the IPM is rewritten into several conflict-free sub-IPMs [2, 4, 14]. The "avoid" method avoids selecting conflicting combinations, used for example in the AETG system [2].

This study introduces two additional conflict handling methods plus a simple method for reducing the size of the final test suite. The study also contains an experiment in which the four conflict handling methods are applied to a number of IPMs with conflicts and the sizes of the resulting conflict-free test suites are compared. The test suite reduction method can be used together with three of the four conflict handling methods to form three new conflict handling methods. In total, this paper investigates seven conflict handling methods.

Very little is known about conflicts and their distribution in practice. For now, we assume that conflicts are more likely to be few and small rather than many and large. This assumption is based on the observation that input parameter models with many conflicts are both difficult to build, understand, and use in subsequent test case generation steps.

The remainder of this paper is structured as follows. Section 2 presents the investigated conflict handling methods in more detail. Section 3 describes the experimental set-up and the variables in the experiment. Section 4 presents results and section 5 contains a summary and some general conclusions.

2. Conflict Handling Strategies

The investigated conflict handling methods are based on four different ideas, two of which are introduced in this paper. The (i) *abstract parameter* (new to this paper) and (ii) *sub-models* [2, 4, 14] methods result in conflict-free input parameter models. The (iii) *avoid* method [2] guarantees that only conflict-free test cases are selected. Finally, the (iv) *replace* (new to this paper) method removes conflicts from already selected test cases. The following subsections describe these conflict handling methods in more detail.

2.1. Abstract Parameters

The *abstract parameter* method introduces new parameters that are abstract representations of two or more original parameters. The values of the abstract parameter consist of conflict-free sub-combinations of the replaced parameters such that the desired coverage is satisfied. Figure 2 shows an algorithm to identify the values of an abstract parameter. First the conflict to be resolved is selected. Then, the parameters involved in the conflict are identified. In the third step, all possible combinations of values of the conflicting parameters are generated. Any combination containing a conflict is removed in the fourth step. Depending on the relation between the number of parameters involved in the conflict and the desired coverage, the set of combinations may be reduced with maintained coverage. The reduction algorithm is omitted for space reasons but the ideas covered in section 2.5 can be applied as well.

When the values of the abstract parameter are decided, the abstract parameter is combined with the remaining pa-

1. Choose the conflict to resolve
2. Identify the i parameters involved in the conflict
3. Generate all i-tuples of values of the i parameters
4. Remove all tuples with conflicts
5. If $i \leq t$
   stop
else
   remove combinations that do not increase coverage

Figure 2. Abstract parameter algorithm for t-wise coverage.

rameters from the original IPM to form a new conflict-free IPM. The test cases generated are transformed back to the values of the original parameters in a post-processing step.

Consider the example in Figure 1, which has conflicts involving parameters A and B. These are replaced by the abstract parameter AB. The values of AB depend on the coverage criterion being used. In 1-wise coverage each parameter value has to be included in at least one test case. This can be achieved if $AB_1 = \{(A1,\ B1), (A2,\ B2), (A3,\ B3)\}$. The complete input parameter model for 1-wise coverage is shown in Figure 3.

$\mathbf{AB_1}$(Size, Occurs) = $\{\mathbf{1:}(0, \text{No}), \mathbf{2:}(1, \text{Once}),$
$\qquad\qquad\qquad\qquad \mathbf{3:}(>1, \text{More than once})\}$
$\mathbf{C}$ (Sorting) = $\{\mathbf{1:}\text{Unsorted}, \mathbf{2:}\text{Sorted}, \mathbf{3:}\text{All identical}\}$

Figure 3. Conflict-free IPM for find function by abstract parameters for 1-wise coverage.

For 2-wise coverage all possible conflict-free pairs must be included, which can be satisfied if $AB_2 = \{(A1,\ B1), (A2,\ B1), (A2,\ B2), (A3,\ B1), (A3,\ B2), (A3,\ B3)\}$.

For t-wise coverage, where t is greater than 1, the final test suite may be unnecessarily large. The reason is that the abstract parameter method will lead to over-representation of some sub-combinations. Consider for instance the IPM for 2-wise coverage of the find function. To satisfy 2-wise coverage, value 1 of parameter AB_2 has to be combined with all three values of parameter C. The result after transformation back to the original IPM parameters A, B and C, the pair $(A1,\ B1)$ occurs at least three times in the final test suite, even though one occurrence is enough to satisfy the coverage criterion. In the general case, there may be complete test cases that do not contribute to coverage and hence could be removed. Section 2.5 describes how test suites are reduced and section 4.1 contains results both with and without reduction.

2.2. Sub-models

The *sub-model* method was first sketched by Williams and Probert [14], and later mentioned by Cohen et al. [2]. Neither paper fully describes the method, thus the following description includes some refinement.

As in the abstract parameter method, the *sub-model* method removes the conflicts from the IPM. An input parameter model with conflicts is rewritten into two or more smaller conflict-free IPMs. Test suites are then generated for each input parameter model and the final test suite is constructed by taking the union of the test suites.

To construct the conflict-free sub-models, the first step is to select a split parameter. The *split parameter* is the parameter involved in a conflict that has the least number of values. In the second step, the IPM is *split* into intermediate sub-models, one for each value of the split parameter. Each of these sub-models contains one unique value of the split parameter and all the values of every other parameter. In the third step, the intermediate sub-models that contain conflicts involving the value of the split parameter are further developed by *removing* values of the other parameters to eliminate the conflicts.

If other conflicts still remain in the intermediate sub-models the method is applied recursively. When all sub-models are conflict-free, some sub-models can be merged. A merge is possible if two sub-models differ in only one parameter.

When no more merges can be done, test cases are generated for each sub-model. The final test suite is the union of the test suites from each sub-model.

Consider the conflict example in Figure 1. Both A and B contain the same number of values, so either can be used as the split parameter. Suppose parameter B is selected. The IPM is split into three sub-IPMs, one for each value of B. These three sub-IPMs are shown in Figure 4.

After the split, sub-IPM 2 contains one conflict $(A1, B2)$ and sub-IPM 3 contains two conflicts $(A1, B3)$ and $(A2, B3)$. These conflicts are eliminated by removing values of A in the respective sub-IPMs. The final conflict-free result is shown in Figure 5. In this example all three sub-models differ in both parameters A and C so merging is not possible.

As in the case of the abstract parameter method, the sub-model method may result in a final test suite that is unnecessarily large. All values of every parameter not involved in any conflicts are included in every sub-model, which may result in partly overlapping test cases. Consider 2-wise coverage of the sub-models in Figure 5. All three sub-models contain the values $A3$ and $C3$. This pair occurs at least three times in different test cases in the final test suite, even though one occurrence is enough to satisfy 2-wise coverage. Thus, test cases may exist that do not contribute to the cov-

sub-IPM 1
A (Size of vector) = {**1:**0, **2:**1, **3:**>1}
B (Element occurs) = {**1:**No}
C (Sorting) = {**1:**Unsorted, **2:**Sorted, **3:**All identical}

sub-IPM 2 **A** (Size of vector) = {**1:**0, **2:**1, **3:**>1}
B (Element occurs) = {**2:**Once}
C (Sorting) = {**1:**Unsorted, **2:**Sorted, **3:**All identical}

sub-IPM 3
A (Size of vector) = {**1:**0, **2:**1, **3:**>1}
B (Element occurs) = {**3:**More than once}
C (Sorting) = {**1:**Unsorted, **2:**Sorted, **3:**All identical}

Figure 4. Intermediate sub-IPMs of find function.

sub-IPM 1
A (Size of vector) = {**1:**0, **2:**1, **3:**>1}
B (Element occurs) = {**1:**No}
C (Sorting) = {**1:**Unsorted, **2:**Sorted, **3:**All identical}

sub-IPM 2'
A (Size of vector) = {**2:**1, **3:**>1}
B (Element occurs) = {**2:**Once}
C (Sorting) = {**1:**Unsorted, **2:**Sorted, **3:**All identical}

sub-IPM 3'
A (Size of vector) = {**3:**>1}
B (Element occurs) = {**3:**More than once}
C (Sorting) = {**1:**Unsorted, **2:**Sorted, **3:**All identical}

Figure 5. Final conflict-free set of sub-IPMs of find function.

erage. Section 4.1 contains results both with and without reduction.

2.3. Avoid Conflicting Test Cases

Cohen et al. describe a conflict handling method that is integrated in the test case selection algorithm used by AETG [2]. The *avoid* method prohibits the combination strategy from selecting any test case that includes conflicting sub-combinations. Instead the combination strategy is forced to find conflict-free test cases until the desired coverage criterion is reached. A similar idea was used by Ostrand and Balcer in the category partition method [12]. In their approach every conflict-free combination was selected.

Consider 1-wise coverage of the find function example in Figure 1. Assume that test cases $(A1, B1, C1)$ and $(A3, B2, C2)$ have already been selected. Without con-

sidering any conflicts, the final test case would then be $(A2, B3, C3)$. However, this test case contains the conflict $(A2, B3)$. With the avoid method integrated in the combination strategy, after $A2$ has been selected the combination strategy is forced to select $B1$ or $B2$ to avoid the conflict.

The general approach of the avoid method is to select the next unused value that will not cause a conflict. If no conflict-free unused values exist, an already used conflict-free value must be used.

The applicability of this conflict handling method is limited since it only can be used with combination strategies that generate their test cases step-by-step. Orthogonal arrays [11] is an example of a combination strategy that cannot use this conflict handling method since the algorithm works with sets of test cases as its smallest entity.

Test suites produced by the avoid method cannot be reduced since each test case is guaranteed to contribute to the coverage.

2.4. Replacing Conflicting Test Cases

Our second suggestion for handling conflicts is the *replace* method. The idea is to resolve conflicts after the test suite has been generated while preserving the coverage.

The naive approach is to remove all test cases with conflicts. However, this would violate the coverage preserving principle since the test cases removed may contain other parts that contribute to the coverage. Instead, each conflicting test case is *cloned*. In each clone one or more of the values involved in the conflict are changed to an arbitrary value that removes the conflict. The number of clones is chosen to preserve the coverage.

If several conflicts affect the same test case, conflicts are removed one at a time via cloning, which guarantees termination.

Table 1 shows a test suite that satisfies 2-wise coverage for the find function test problem without regards to the conflicts. The resulting test suite has three test cases with conflicts: $TC2$, $TC3$, and $TC6$.

Pair-wise coverage and conflicts that involve two parameters mean two clones are created for each of the three conflicting test cases. The value of the first parameter in the conflict is changed in the first clone and the value of the second parameter in the conflict is changed in the second clone of each pair of clones. This is shown in Table 2.

With the replace method, the resulting test suite may be unnecessarily large since the clones overlap. This means that some sub-combinations of values are included more than once. Section 4.1 contains results both with and without reduction.

4

Table 1. Test suite for 2-wise coverage for the find function, conflicts highlighted.

TC	Parameters		
	A	B	C
TC1	1	1	1
TC2	**1**	**2**	2
TC3	**1**	**3**	3
TC4	2	1	2
TC5	2	2	3
TC6	**2**	**3**	1
TC7	3	1	3
TC8	3	2	1
TC9	3	3	2

Table 2. The two steps to remove conflicts from the find function example.

Step 1 - Clone				Step 2 - Replace		
	Parameters			Parameters		
TC	A	B	C	A	B	C
TC1	1	1	1	1	1	1
TC2a	*	2	2	**2**	2	2
TC2b	1	*	2	1	**1**	2
TC3a	*	3	3	**3**	3	3
TC3b	1	*	3	1	**1**	3
TC4	2	1	2	2	1	2
TC5	2	2	3	2	2	3
TC6a	*	3	1	**3**	3	1
TC6b	2	*	1	2	**1**	1
TC7	3	1	3	3	1	3
TC8	3	2	1	3	2	1
TC9	3	3	2	3	3	2

2.5. Reducing the Test Suite

As explained in the previous subsections, three of the four investigated conflict handling methods, abstract parameters, sub-models, and replace, may result in unnecessarily large test suites. Thus, we included a mechanism to reduce the size of the test suite.

Finding a minimal set of combinations that satisfy a given level of coverage is an NP-complete problem. Hence, any tractable solution has to rely on heuristics. The focus of this research is not on finding the best reduction algorithm, instead we selected a very simple scheme that was easy to implement and efficient to execute.

The reduction algorithm used in this experiment examines one test case at a time. If the current test case increases coverage it is kept, otherwise it is discarded. Hence, a possibly smaller test suite with the same level of coverage as the original test suite is produced.

3. Experiment Description

The goal of this experiment is to explore the performance of the seven conflict handling methods. As described in section 1, all methods preserve the initial level of coverage. This means that they can be considered equally effective. Hence, the focus of this experiment is on the efficiency of the conflict handling methods. Even if different conflict handling methods may have different set-up costs, the dominating cost will, at some point, be related to the size of the final test suite. An important part of the reason is that identification of expected results for each test case is likely to require human intervention.

Based on this reasoning, the selected response (dependent) variable for this experiment is the size of the test suite, as measured by the number of test cases. The controlled (independent) variables in this experiment are *size* of the input parameter model (with respect to number of parameters and values), *coverage criterion* used, and *number* and *type* of conflicts. Each combination of values of the dependent variables is a unique test problem. All 864 possible test problems are included in the experiment. Subsections 3.1 - 3.3 describe the specific values of the controlled variables.

In addition to applying the seven conflict handling methods to each test problem, base test suites are created for each IPM size and each level of coverage. No conflicts are used in the base test problems.

The research question formulated for this experiment is: *Does method X generate fewer test cases than the base test suite?*. To answer this question the following null hypothesis is formulated: $\mathbf{H_0}$: *There is no difference in the sizes of the test suites for method X and the base.*

Figure 6 shows an overview of the steps of the experiment. The experiment started with manual preparations of an IPM *base* and a set of conflict files. The IPM base is a description of the IPM sizes that should be used in the experiment. Each conflict file contains the sub-combinations of parameter values that should not be allowed when that conflict file is used.

Then, IPMs are automatically generated from the IPM base. The same IPMs are used by all conflict handling methods. For the abstract parameter and sub-models methods the IPMs are transformed accordingly. These tasks are fully automated for the abstract parameter method and semi-automated for the sub-model method. Test suites are then automatically generated for all four methods. The same generation mechanism is used in all four methods with the addition of a conflict avoidance mechanism for the avoid method.

At this point, conflicts only remain in one set of test

Figure 6. Overview of how the seven conflict handling methods are applied to a set of conflicts.

suites. An automated replace mechanism removes these conflicts. Finally the same, automated, reduction mechanism is executed for the three methods in which it may have an effect. All programs used in this experiment are implemented in Perl.

3.1. Input Parameter Models

The input parameter model represents the input space and possibly other properties, such as state, of the test problem. Malayia [10] and Chen et al. [1] provide practical advice for input parameter modeling, but the best way to create an input parameter model is still an open question.

Based on the input parameter model the combination strategy selects test cases until the desired coverage is achieved irrespective of what the input parameter model actually represents. Hence, size rather than contents of the input parameter model is the main influencing factor on the size of the test suite.

The size of the input parameter model is defined by the number of parameters and the number of values of each parameter. Thus, it is interesting to include several input parameter models with different sizes in an experiment such as this. We decided to use all combinations of $5, 7, 9,$ and 11 parameters with $4, 5, 6, 7, 8,$ and 9 values, i.e., 24 combinations. These sizes are likely to exist in real testing problems although it is also likely that other, larger, combinations exist.

Similar sizes of input parameter models have been used in previous experiments [3, 5, 8, 13]. However, the main reason for our choice of IPM sizes is time consumption of the experiment. The number of values of the largest parameter has a big impact on the number of test cases of the final test suite for all coverage criteria. In the abstract parameter method, the abstract parameter may contain as many values as the product of the number of values of the original parameters. Hence nine parameter values of two parameters involved in a conflict still produces an abstract parameter with 80 values in the worst case. The 81st pair is impossible and thus not included.

For each combination of number of parameters and parameter values, the IPM can either be unbalanced or balanced. In an unbalanced IPM, parameters have different numbers of values, whereas in balanced IPMs, all parameters have the same number of values. Unbalanced IPMs are more natural. Balanced IPMs give results that are easier to analyze since there is one fewer factor that can influence the results.

The interpretation of an unbalanced IPM with V values is that at least one parameter has V values and the other parameters have between two and V values, with at least one parameter having less than V values. With this interpretation it is possible to create many different unbalanced IPMs with the same number of parameters and parameter values. In this experiment we decided to generate one unbalanced IPM for each combination number of parameters and number of values. To our knowledge, characteristics of IPMs, such as the number of values of different IPM parameters in practice, is not known; hence we opted for randomly assigned sizes to get diversity.

To summarize, this experiment was conducted with 48 different IPMs ranging in size from an IPM with five pa-

rameters with at most four values each up to an IPM with eleven parameters each with nine values.

3.2. Coverage Criteria

Section 2.1 shows that some of the conflict handling methods investigated need to be tailored for specific coverage criteria. Thus, it is natural to investigate what happens with different criteria. This experiment used three coverage criteria, 1-wise, 2-wise, and 3-wise coverage, as defined in section 1.

3.3. Conflicts

Very little is known about conflicts and their distribution in practice. Our assumption is that conflicts tend to be few and small rather than many or involving many parameters. One important basis for this assumption is the more conflicts the harder to analyze the IPM with respect to correctness. Despite this assumption this experiment was executed with conflict sizes ranging from small to rather big with respect to the IPM sizes.

Altogether, six different sets of conflicts were used in this experiment as summarized in Table 3.

Table 3. Overview of conflicts used in this experiment

Conflict Set	Number of involved parameters	Number of conflicting pairs
CS1	2	1
CS2	2	2
CS3	2	2
CS4	4	3
CS5	4	3
CS6	5	10

The pairs of conflict sets $(CS2, CS3)$ and $(CS4, CS5)$, differ with respect to how the conflict pairs in the set are distributed across the IPM parameters. For instance, in $CS4$, no parameter is involved in more than two conflict pairs, whereas in $CS5$ one parameter is involved in all three conflict pairs.

3.4. Threats to Validity

It is a potential threat to the internal validity that the same persons both propose and then evaluate some of the methods. In this experiment deliberate measures have been taken not to favor these methods. The conflicts used in this experiment introduce a slight bias to the avoid method over

the replace and reduced replace methods. The reason is that all conflicts used in this experiment were selected such that they were included in the final test suites. Suppose that the test suite $\{(A1, B1, C1), (A2, B2, C2), (A3, B3, C3)\}$ yield 1-wise coverage for an IPM without conflicts. A conflict between $A1$ and $B1$ requires $(A1, B1, C1)$ to be replaced by two test cases while a conflict between $A1$ and $B2$ does not affect the size of the test suite for the replace method.

A more severe threat to the internal validity is the use of random numbers in the implementations of the AETG method [2]; they are used to generate test suites with 2-wise and 3-wise coverage. The heuristic nature of AETG may cause different test suites to be generated for the same test problem. A few test suites in this experiment contain more or fewer test cases compared to the rest, which is probably due to this heuristic property.

This experiment compensates for this non-determinism by sampling multiple test suites for AETG. The comparison of the methods are based on 144 samples for each method.

A common problem for external validity is representativity [9]. In the context of this experiment, neither types nor sizes of conflicts in input parameter models nor actual sizes of IPMs in practice have been extensively studied. Thus, it is difficult to assess the representativity of results of this study.

The last threat to external validity is lack of complete descriptions of some of the conflict handling methods. The sub-model conflict handling method is sketched in a couple of papers [14, 2], but they do not fully describe the method. Our guesses of the details of the method are marked so persons with the proper knowledge can make their own evaluations. A similar situation also arose during the implementation of the AETG combination strategy for 3-wise coverage. In addition to the implementation used in this experiment there is at least one alternative way to implement the algorithm. We try to clearly present the selected interpretation [7].

4. Results

Subsection 4.1 contains the quantitative results of the experiment. The assumption of this experiment is that the number of test cases in the final test suites is the influencing factor when selecting a conflict handling method to use. However, other factors, such as total time consumption of method and ease of use, may also influence this choice. These factors are discussed in subsection 4.2.

4.1. Test Suite Sizes

Figures 7 and 8 show box-plots of the number of test cases after applying the seven conflict handling methods us-
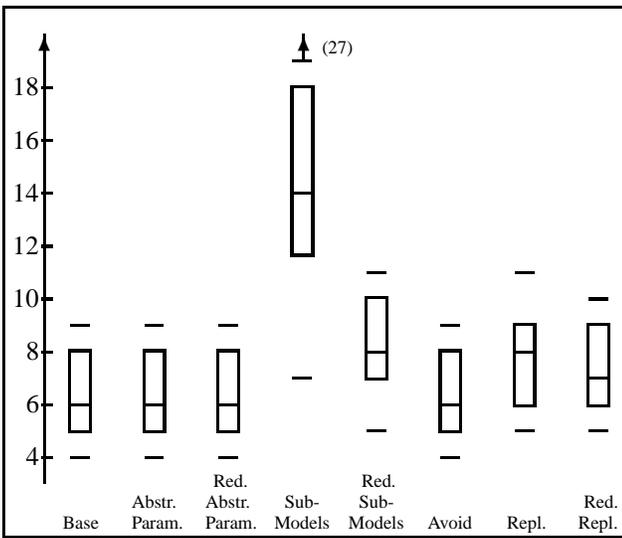
Figure 7. Box-plots of the sizes of test suites for 1-wise coverage using small conflict sets.
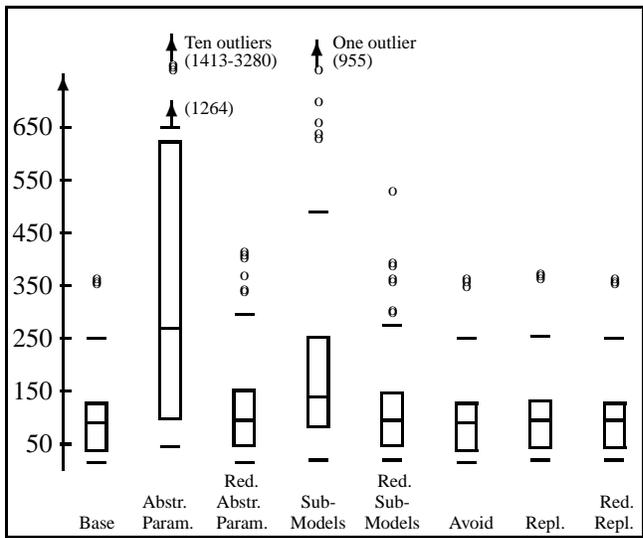

Figure 8. Box-plots of the sizes of test suites for 2-wise coverage using small conflict sets.

ing the three small conflict sets (CS1, CS2, and CS3) with 1-wise and 2-wise coverage.

A box-plot summarizes the distribution of a set of values. The box represents the middle 50% of the values. The line in the middle of the box marks the median. The bar above the box is called the *upper adjacent value* and it marks the largest value in the set smaller than the upper fence. The *upper fence*, in turn, is the top value of the box plus 1.5 times the distance between the floor and the ceiling of the box. The lower adjacent value is derived correspondingly from the bottom value of the box and is marked with a bar under the box. Values beyond the adjacent values, if there are any, are called *outliers* and are marked with rings.

Each box-plot represents sizes of 144 test suites. These test suites are the results from taking the same 48 IPMs augmented with the three different conflict sets CS1, CS2, and CS3.

For 1-wise coverage of the IPMs with the small conflict sets, both Chi-square and F-tests show with 0.95% probability that the Null-hypothesis **cannot** be rejected for all methods except the sub-models method.

In a conflict-free test suite with 2-wise coverage, every pair of values of any two parameters are included in the test suite. This means that a test suite must always contain at least as many test cases as the product of the number of values of the two largest parameters. A conflict between two parameter values means that this pair of values should not be included in the test suite. Thus, if the two largest parameters are involved in the conflict there is a chance that the number of test cases in the conflict-free test suite is actually lower than the corresponding base case.

As with 1-wise coverage, the avoid, replace and reduced replace methods have almost the same number of tests as the base case does. For these three methods both Chi-square and F-tests show with 0.95% probability that the null-hypothesis **cannot** be rejected. For the other four methods the null-hypothesis is rejected.

The results for 3-wise coverage conform to the results for 2-wise coverage. These results are omitted for space reasons but can be found in the companion technical report [7].

Based on the results from using the small conflict sets (CS1, CS2, and CS3) the experiment was optimized with respect to experiment execution time before applying the larger conflict sets (CS4, CS5, and CS6). The optimizations included using only half of the IPMs compared with the small conflict sets and completely omitting the abstract parameter method. Two factors contributed to the decision to drop the abstract parameter method. The first was that the results from applying the abstract parameter methods to the small conflict sets indicates that this method is less efficient than the other methods. The second was that the size of the abstract parameter is close to the product of the sizes of the original parameters, which has a profound effect on the generation times of the test suites.

Table 4 shows, for each coverage criterion and each conflict set, the percentage change in the sum of the test cases over all IPMs compared with the corresponding test cases for the conflict-free base cases.

The previous observation that the avoid method behaves similarly as the base case also holds for the larger conflict sets (CS4, CS5, and CS6). In contrast, both the replace and the reduced replace methods perform worse with the larger

Table 4. Percentage change of the sum of test cases for all IPMs w.r.t. the corresponding base cases.

x-wise coverage		Conflict Set (CS)					
		1	2	3	4	5	6
Abst. Param.	1	15	31	31	-	-	-
	2	398	399	379	-	-	-
	3	-	-	-	-	-	-
Red. Abst. Param.	1	15	31	31	-	-	-
	2	29	26	26	-	-	-
	3	-	-	-	-	-	-
Sub-Models	1	99	99	189	494	296	1219
	2	83	79	160	361	242	778
	3	80	80	166	335	227	687
Red. Sub-Models	1	15	31	29	109	42	211
	2	6	14	50	84	34	190
	3	8	21	17	106	43	253
Avoid	1	0	0	0	0	0	0
	2	0	-2	-2	-2	-2	-5
	3	0	0	-1	-1	-1	-1
Repl.	1	15	15	31	74	19	249
	2	3	5	5	11	10	32
	3	3	5	5	10	9	29
Red. Repl.	1	15	15	15	35	18	68
	2	1	2	2	4	5	14
	3	2	4	4	7	6	16

conflict sets. Finally, the sub-model and reduced sub-model methods perform even more poorly with respect to the number of test cases. Taking the larger conflict sets into account, Chi-square and F-tests show with 0.95% probability that the null-hypothesis **cannot** be rejected for the avoid method but rejects the null-hypothesis for the other four methods.

To summarize our findings with respect to test suite sizes, the avoid method performs best across the whole experiment. In most cases it is possible to see a slight decrease in the number of test cases compared to the base case. The reduced replace method also performs similarly to the base case for small conflict sets. As the conflict sets grow the performance of the reduced replace method deteriorates slightly. The other five methods performs significantly worse.

4.2. Discussion

In addition to the size of the test suite, other properties of the conflict handling methods may also influence the tester's choice. Four such properties are automation, time consumption, ease of use, and applicability of the conflict handling methods. Because these properties heavily depend on the specific automated tool used, an experimental evaluation on our research prototype software would be essentially meaningless. The following paragraphs offer some analytical discussion of these properties.

The avoid and the replace methods are easy to fully automate, although the avoid method must be adapted to each combination strategy. The abstract parameter and sub-model methods are more difficult to automate since they require transformations of the IPM.

When the test suites are generated, with the exception of the abstract parameter method, time is dominated by the test case generation, not conflict handling. This is not true for the abstract parameter method because the original IPM is transformed to a larger IPM due to the abstract parameter.

With regards to ease of use, the avoid and replace methods should only require a command to be invoked. The abstract parameter and sub-model methods require the input parameter model to be adjusted, a step that will probably require some human intervention.

The abstract parameter, sub-models, and replace methods are all completely general with respect to the combination strategy used. The avoid method has to be integrated into the combination strategy, which may not be possible for some combination strategies. The orthogonal array method [14] cannot be used with the avoid method.

Apart from an observation by Cohen et al. [2], little other research is available on this topic. They observed without giving all the details that the avoid method was around one order of magnitude better with respect to test suite size than the sub-models method. Their paper did not include enough details for a full comparison. However, the results of this experiment partly agree with their observations although the differences are smaller than one order of magnitude.

5. Conclusions and Future Research

The results of the experiment are important contributions of this experiment. Further, a description of the general problem of conflicts in the IPM and an experimental method to compare different methods to handle such conflicts are also essential contributions.

In this experiment the avoid method stands out as the best conflict handling method. Not only does it consistently produce the smallest test suites, our analysis indicates that it is also easier to use and once implemented, cheaper in terms of time consumption.

Although it is always difficult to generalize results from experiments it is hard to imagine any circumstances that would cause the sub-models and abstract parameter methods to outperform the avoid and replace methods. Other reduction algorithms may result in fewer test cases than the current one so the reduced versions of the sub-models and abstract parameter methods may still be viable options from

the test suite size point of view. However, taking the necessary manual intervention of the sub-models and abstract parameter methods into account, it is still unlikely that these methods would be more efficient than the avoid method.

An important consequence of these results is that the tester does not need to worry about introducing conflicts when creating the input parameter model. As long as the tester keeps track of the conflicting sub-combinations, these can be resolved automatically and efficiently at a later stage.

Our results also suggest that, in general, the avoid method is better than the replace and reduced replace methods. Even if it can be shown that avoid is superior with respect to the size of the test suite there are still times when the avoid method cannot be used. First, there may be a situation where the test suite is already generated and conflicts are added later. Second and perhaps more important is that the avoid strategy requires the source code of the combination strategy to be available. Hence, the use of third party combination strategy products may prevent the avoid strategy from being used. In both these cases the replace and reduced replace methods will work.

At some point, it is likely that the effect of the choice of conflict handling method decreases with higher coverage. When using N-wise coverage, there is only a single set of combinations that satisfies this goal, i.e., every conflict-free combination. Thus it does not matter which strategy is used, they will yield exactly the same result. Finding this point is a topic for further research.

Another line of research is to investigate the effects of more and larger conflicts in the IPM. It would be very interesting to investigate to what extent conflicts exist and what they look like in IPMs in practice. This is particularly interesting for researchers looking into the problem of input parameter modeling.

The effects of different reduction algorithms would also be interesting to investigate.

6. Acknowledgments

References

[1] T. Chen, P.-L. Poon, S.-F. Tang, and T. Tse. On the Identification of Categories and Choices for Specification-based Test Case Generation. *Information and Software Technology*, 46(13):887–898, 2004.

[2] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG System: An Approach to Testing Based on Combinatorial Design. *IEEE Transactions on Software Engineering*, 23(7):437–444, July 1997.

[3] M. Cohen, P. Gibbons, W. Mugridge, and C. Colburn. Constructing test cases for interaction testing. In *Proceedings of the 25th International Conference on Software Engineering, (ICSE'03), Portland, Oregon, USA, May 3-10, 2003*, pages 38–48. IEEE Computer Society, May 2003.

[4] N. Daley, D. Hoffman, and P. Strooper. A framework for table driven testing of Java classes. *Software - Practice and Experience (SP&E)*, 32(5):465–493, April 2002.

[5] M. Grindal, B. Lindström, A. J. Offutt, and S. F. Andler. An evaluation of combination strategies for test case selection. *Empirical Software Engineering*, Available in SpringerLink electronic Online First version, 2006.

[6] M. Grindal, A. J. Offutt, and S. F. Andler. Combination testing strategies: A survey. *Software Testing, Verification, and Reliability*, 15(3):167–199, September 2005.

[7] M. Grindal, A. J. Offutt, and J. Mellin. Handling Constraints in the Input Space when Using Combination Strategies for Software Testing. Technical Report HS-IKI-TR-06-001, School of Humanities and Informatics, University of Skövde, 2006.

[8] Y. Lei and K. C. Tai. In-parameter-order: A test generation strategy for pair-wise testing. In *Proceedings of the third IEEE High Assurance Systems Engineering Symposium*, pages 254–261. IEEE, Nov. 1998.

[9] B. Lindström, M. Grindal, and A. J. Offutt. Using an existing suite of test objects: Experience from a testing experiment. *ACM SIGSOFT Software Engineering Notes, SECTION: Workshop on empirical research in software testing papers, Boston*, 29(5), Nov. 2004.

[10] Y. K. Malaiya. Antirandom testing: Getting the most out of black-box testing. In *Proceedings of the International Symposium On Software Reliability Engineering, (ISSRE'95), Toulouse, France, Oct, 1995*, pages 86–95, Oct. 1995.

[11] R. Mandl. Orthogonal Latin Squares: An application of experiment design to compiler testing. *Communications of the ACM*, 28(10):1054–1058, October 1985.

[12] T. J. Ostrand and M. J. Balcer. The Category-Partition Method for Specifying and Generating Functional Tests. *Communications of the ACM*, 31(6):676–686, June 1988.

[13] T. Shiba, T. Tsuchiya, and T. Kikuno. Using artificial life techniques to generate test cases for combinatorial testing. In *Proceedings of 28th Annual International Computer Software and Applications Conference (COMPSAC'04) 2004, Hong Kong, China, 28-30 September 2004*, pages 72–77. IEEE Computer Society, 2004.

[14] A. W. Williams and R. L. Probert. A practical strategy for testing pair-wise coverage of network interfaces. In *Proceedings of the 7th International Symposium on Software Reliability Engineering (ISSRE96), White Plains, New York, USA, Oct 30 - Nov 2, 1996*, pages 246–254, Nov 1996.