Handling Constraints in the Input Space when Using Combination Strategies for Software Testing

Mats Grindal*, Jeff Offutt† and Jonas Mellin‡

2006-01-27

Technical Report HS-IKI-TR-06-001
School of Humanities and Informatics
University of Skövde

Abstract

This study compares seven different methods for handling constraints in input parameter models when using combination strategies to select test cases. Combination strategies are used to select test cases based on input parameter models. An input parameter model is a representation of the input space of the system under test via a set of parameters and values for these parameters. A test case is one specific combination of values for all the parameters.

Sometimes the input parameter model may contain parameters that are not independent. Some sub-combinations of values of the dependent parameters may not be valid, i.e., these sub-combinations do not make sense. Combination strategies, in their basic forms, do not take into account any semantic information. Thus, invalid sub-combinations may be included in test cases in the test suite.

This paper proposes four new constraint handling methods and compares these with three existing methods in an experiment in which the seven constraint handling methods are used to handle a number of different constraints in different sized input parameter models under three different coverage criteria. All in all, 2568 test suites with a total of 634263 test cases have been generated within the scope of this experiment.

Keywords: Constraint Handling, Category Partition, AETG, Test Case Selection, Input Parameter Modeling

*School of Humanities and Informatics, University of Skövde, mats.grindal@his.se
†Information and Software Engineering, George Mason University, Fairfax, VA 22030, USA, offutt@ise.gmu.edu
‡School of Humanities and Informatics, University of Skövde, jonas.mellin@his.se

1 Introduction

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

A large number of combination strategies have been proposed and are surveyed in our previous paper [GOA05]. A common property of all combination strategies is that they all require an input parameter model. The *input parameter model* (IPM) is a model of the input space of the test object. The IPM may either be based on the actual input parameters of the test object or it may be based on some abstract parameters, which in turn are functions of the actual parameters. The IPM lists the parameters and a set of representative values selected by the tester for each parameter. The category partition method [OB88] is a structured way of creating an input parameter model.

Based on the IPM, combination strategies generate test cases by selecting one value for each parameter and combining these into complete 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. 100% 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 often covers 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. t-wise coverage is formally defined by Williams and Probert [WP01].

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

In some cases there will be conflicts built into the IPM. A conflict exists when the result of combining two or more values of different parameters does not make sense. Consider for example a checking that a date is valid. Figure 1 shows an input parameter model for a valid date.

The three parameters of the input parameter model are YYYY, MM, and DD. For the YYYY and DD parameters selection of some representative values has reduced the input spaces. The constraints describe which combinations of parameter values are valid. A number of invalid combinations exist, for instance November 31st and February 29th 2001 are invalid.

In their basic forms, combination strategies generate test suites that satisfy the desired coverage without using any semantic information. Hence, invalid test cases may be selected as part of

(YYYY) = {1900, 1999, 2000, 2001, 2002, 2004}
(MM) = {Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec}
(DD) = {01,28,29,30,31}
Constraints: Apr, Jun, Sep, Nov have maximum 30 days
Feb has maximum 29 days in leap years
Feb has maximum 28 days in other years
All other months have maximum 31 days

Figure 1: Input parameter model for a valid date.

the test suite. How to handle these conflicts has not previously been adequately investigated. Important questions are which constraint handling methods work for which combination strategies and how the size of the test suite is affected by the constraint handling method.

Two methods to handle this situation have been previously proposed. In one method the IPM is rewritten into several conflict-free sub-IPMs [CDFP97, WP96, DHS02]. The other method is to avoid selecting invalid combinations, used for example in the AETG system [CDFP97].

This study describes two additional constraint 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 constraint 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 constraint handling methods forming three new constraint handling methods. In total this paper investigates seven constraint handling methods.

Very little is known about conflicts and their distribution in practice. However, our assumption is that it is more likely for conflicts in an input parameter models to be few and small rather than many and large. This assumption is based on the observation that an input parameter model with many conflicts is both difficult to understand and to use in subsequent test case generation steps. Hence this study is focused on input parameter models containing few and small conflicts.

Figure 2 shows an IPM for a web-based hotel search application, which will be used as a running example in the remainder of this paper. In this system, the user enters her preferences and the system suggests a suitable hotel. Some of the things the user may enter are destination, type of room, a favored activity which should be located close to the hotel, and a cost level. Three conflicts are included in the example; in Aspen there is no beach, and there is no skiing in Miami nor in Los Angeles.

Each parameter value of the IPM is given a unique logical name by assigning unique letters to the parameters and assigning unique values to the different values of each parameter. This logical representation is used to make the implementations of the combination strategies completely general. A consequence of this representation is that constraints can be described as invalid sub-combinations of parameter values. This made also the implementations of the constraint handling

A (Destination) = {**1:**Aspen, **2:**Miami, **3:**Los Angeles}
B (Type of room) = {**1:**Single, **2:**Double, **3:**Suite}
C (Activity) = {**1:**Beach, **2:**Skiing, **3:**Shopping}
D (Cost Level) = {**1:**Budget, **2:**Bronze, **3:**Silver, **4:**Gold }
Invalid sub-combinations: (A1,C1), (A2,C2), (A3,C2)

Figure 2: Partial input parameter model of hotel guide system.

methods general. Functions or relations, often used as compact ways of representing a constraint, can always be translated into lists of invalid sub-combinations.

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

2 Constraint Handling Strategies

In this study we have investigated four different constraint handling methods. The *abstract parameter* and *sub-models* methods result in conflict-free input parameter models. The *avoid* method makes sure that only conflict-free test cases are selected during the test case selection process. Finally, the *replace* method removes conflicts from already selected test cases.

Sometimes it is possible to reduce the size of the final test suite with maintained coverage after the constraint handling method has been applied. This study includes the results of both regular and reduced test suites wherever applicable. In the following subsections each of the four constraint handling methods and the test suite reduction method are described in detail.

2.1 Conflict-free Input Using Abstract Parameters

Our first suggestion for handing constraints is to use abstract parameters. In the *abstract parameter* method, conflicts are removed during the construction of the input parameter model. The main idea is to use one or more abstract parameters to represent valid sub-combinations. The first step of the abstract parameter method is to identify conflicting parameters in an input parameter model containing conflicts. In the second step, an abstract parameter is used to replace the parameters involved in the conflict. The values of the abstract parameter consist of valid sub-combinations of the replaced parameters such that the desired coverage is satisfied.

The abstract parameter is combined with the remaining parameters from the original input parameter model to form a new conflict-free input parameter model. Test cases are then generated from the new IPM and finally, these are transformed back to the values of the original parameters.

Consider the example in figure 2. Values of parameters A and C are involved in conflicts, which means that they should be replaced by an abstract parameter AC. The values of this parameter depend on the coverage criterion used. In 1-wise coverage each parameter value has to be included in at least one test case. For instance, this is achieved with AC = {(A1, C2), (A2, C1), (A3, C3)}. In 2-wise coverage all possible valid pairs need to be included, which results in AC = {(A1, C2), (A1, C3), (A2, C1), (A2, C3), (A3, C1), (A3, C3)}.

Figures 3 and 4 show the complete input parameter models with abstract parameters for 1 and 2-wise coverage.

AC (Dest, Act) = {**1:**(Aspen, Skiing), **2:**(Miami, Beach), **3:**(Los Angeles, Shopping)}
B (Type of room) = {**1:**Single, **2:**Double, **3:**Suite}
D (Cost Level) = {**1:**Budget, **2:**Bronze, **3:**Silver, **4:**Gold }

Figure 3: Conflict-free input parameter model of hotel guide system using abstract parameters for 1-wise coverage.

AC (Dest, Act) = {**1:**(Aspen, Skiing), **2:**(Aspen, Shopping),
3:(Miami, Beach), **4:**(Miami, Shopping),
5:(Los Angeles, Beach), **6:**(Los Angeles, Shopping)}
B (Type of room) = {**1:**Single, **2:**Double, **3:**Suite}
D (Cost Level) = {**1:**Budget, **2:**Bronze, **3:**Silver, **4:**Gold }

Figure 4: Conflict-free input parameter model of hotel guide system using abstract parameters for 2-wise coverage.

For t-wise coverage where t is greater than 1, it may be the case that the final test suite is unnecessarily large. The reason is that the abstract parameter will lead to over-representation of some sub-combinations. Consider the example in figure 3. To satisfy pair-wise coverage, value 1 of parameter AC has to be combined with all four values of parameter D. Thus, the pair $(Aspen, Skiing)$ will occur at least four times in the final test suite, even though one occurrence is enough to satisfy the coverage criterion. Thus, there may be test cases that do not contribute to coverage and hence could be removed. In this study we have included results both from regular and reduced test suites.

2.2 Conflict-free Input Using Sub-models

In the context of combination strategies, the sub-model scheme was first sketched by Williams and Probert [WP96]. Later it was also mentioned by Cohen et al. [CDFP97]. Neither of these provide

the full details of the method, thus the following description includes some guesswork.

Like in the abstract parameter method, the *sub-model* method removes the conflicts during the construction of the input parameter model. In this method, an input parameter model containing conflicts is rewritten into two or more smaller conflict-free IPMs. Test cases 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 a parameter involved in a conflict with the least number of values. Then, 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 slit parameter are further developed by *removing* values of the other parameters such that the conflicts are eliminated.

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 only in one parameter.

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

Consider the example in figure 2. Values of parameters A and C are involved in the conflict. Both parameters contain the same number of values so either one can be used as the split parameter. Suppose parameter C is selected. The IPM is split into three sub-IPMs, one for each value of C. These three sub-IPMs are shown in figure 5.

Sub-IPMs 1 and 2 still contain conflicts, thus values from parameter A have to be removed in both cases. The final conflict-free result is shown in figure 6. In this example all three sub-models differ in both parameters A and C so merging is not possible.

As with the abstract parameter method, the final test suite may be unnecessarily large. In this case the reason is that all values of every parameter not involved in any conflicts will be included in every sub-model, which may result in partly overlapping test cases. Consider pair-wise coverage of the sub-models in figure 6. All three sub-models contain the values $B1$ and $D1$. This pair will occur at least three times in different test cases in the final test suite, even though one occurrence is enough to satisfy the coverage criterion. Thus, there may be test cases that do not contribute to coverage and hence could be removed. In this study we have included results both from regular and reduced test suites.

Applying the sub-model method is straightforward as long as only two parameters are involved in the conflict but when three or more parameters are involved in a conflict it is not obvious how to apply the method. The main reason is that a three-parameter conflict may be interpreted in more than one way, which will affect this method. Section 3.3 gives further details on this topic.

sub-IPM 1
A (Destination) = {**1:**Aspen, **2:**Miami, **3:**Los Angeles}
B (Type of room) = {**1:**Single, **2:**Double, **3:**Suite}
C (Activity) = {**1:**Beach}
D (Cost Level) = {**1:**Budget, **2:**Bronze, **3:**Silver, **4:**Gold }

sub-IPM 2
A (Destination) = {**1:**Aspen, **2:**Miami, **3:**Los Angeles}
B (Type of room) = {**1:**Single, **2:**Double, **3:**Suite}
C (Activity) = {**2:**Skiing}
D (Cost Level) = {**1:**Budget, **2:**Bronze, **3:**Silver, **4:**Gold }

sub-IPM 3
A (Destination) = {**1:**Aspen, **2:**Miami, **3:**Los Angeles}
B (Type of room) = {**1:**Single, **2:**Double, **3:**Suite}
C (Activity) = {**3:**Shopping}
D (Cost Level) = {**1:**Budget, **2:**Bronze, **3:**Silver, **4:**Gold }

Figure 5: Intermediate sub-IPMs of hotel guide system.

2.3 Avoiding the Selection of Conflicting Test Cases

Cohen et al. described a constraint handling method integrated in the test case selection mechanism, in their case the AETG tool [CDFP97]. The idea in the general form is to supply a list of invalid sub-combinations and prohibit the combination strategy from selecting any test case including these sub-combinations. Instead the combination strategy must find valid test cases until the desired coverage criterion is reached. A similar idea was used by Ostrand and Balcer in the category partition method [OB88]. In their approach every valid combination was selected, so there was no need to keep track of the currently reached coverage.

The applicability of this constraint handling method is limited since it only can be used with combination strategies that build their test suites step-by-step. Orthogonal arrays [Man85] is an example of a combination strategy that cannot use this constraint handling method since all test cases are selected instantly.

Suppose that 1-wise coverage is the goal for the hotel guide system example in figure 6. If conflicts are not considered, the first test case $(A1, B1, C1, D1)$ is the result of combining the first unused value of each parameter. However, this test case contains a conflict $(A1, C1)$. If the avoid method has been integrated in the combination strategy, the first test case would instead be $(A1, B1, C2, D1)$. After $A1$ and $B1$ have been selected, the algorithm would be prevented from

sub-IPM 1'
A (Destination) = {**2:**Miami, **3:**Los Angeles}
B (Type of room) = {**1:**Single, **2:**Double, **3:**Suite}
C (Activity) = {**1:**Beach}
D (Cost Level) = {**1:**Budget, **2:**Bronze, **3:**Silver, **4:**Gold }

sub-IPM 2'
A (Destination) = {**1:**Aspen}
B (Type of room) = {**1:**Single, **2:**Double, **3:**Suite}
C (Activity) = {**2:**Skiing}
D (Cost Level) = {**1:**Budget, **2:**Bronze, **3:**Silver, **4:**Gold }

sub-IPM 3
A (Destination) = {**1:**Aspen, **2:**Miami, **3:**Los Angeles}
B (Type of room) = {**1:**Single, **2:**Double, **3:**Suite}
C (Activity) = {**3:**Shopping}
D (Cost Level) = {**1:**Budget, **2:**Bronze, **3:**Silver, **4:**Gold }

Figure 6: Final conflict-free input parameter model of hotel guide system based on sub-models

selecting $C1$ since it would cause a conflict with one of the already selected parameter values in the test case. Instead the next unused value that will not cause a conflict is used. If there are no more unused values that are conflict-free an already used conflict-free value should be used.

In the avoid method, each new test case is a step towards the final solution since at least one previously unused value or sub-combination. This means that attempts to reduce the size of the test suite will not result in any further gain.

2.4 Replacing Conflicting Test Cases

Our second suggestion for handling constraints is the **replace** method. It builds on the idea that conflicts in test cases can be resolved after the test suite has been generated. A requirement for this method to work is that the method preserves the coverage of the test suite.

The naive approach would be to just dispose of the invalid test cases, but this does not work since this may violate the coverage. Consider test case $(A1, B1, C1)$ in a test suite that satisfies pair-wise coverage. Further, suppose sub-combination $(A1, B1)$ is invalid. If this is the only test case that contains the valid sub-combination $(A1, C1)$, removing the test case would violate the coverage.

Instead, the invalid test case is cloned, in each clone one or more of the parameter values

Test Case	Parameters A	B	C	D
TC1	**1**	1	**1**	1
TC2	1	2	3	3
TC3	**1**	3	**1**	4
TC4	1	2	2	2
TC5	2	1	1	2
TC6	**2**	2	**2**	4
TC7	2	3	1	3
TC8	2	2	3	1
TC9	3	1	3	4
TC10	3	2	1	2
TC11	**3**	3	**2**	1
TC12	**3**	1	**2**	3
TC13	1	3	3	2

Table 1: Test suite satisfying 100% pair-wise coverage for the hotel guide system test problem without considering the conflicts. Conflicts highlighted.

involved in the conflict is changed to an arbitrary value that removes the conflict. The number of clones is chosen such that the coverage is preserved.

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 hotel guide system test problem without regard to the conflicts. In the resulting test suite there are five invalid test cases: $TC1$, $TC3$, $TC6$, $TC11$, and $TC12$.

Pair-wise coverage and conflicts involving two parameters means two clones are created for each of the four 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. The resulting test suite satisfies pair-wise coverage of all *valid* sub-combinations of any two parameters.

With the replace method, the resulting test suite may be unnecessarily large. The reason is that the clones are partly overlapping which means that some sub-combinations of values are included more than once. Thus, there may be test cases that do not contribute to coverage and hence could be removed. In this study we have included results both from regular and reduced test suites.

Step 1 - Clone				
	Parameters			
Test Case	**A**	**B**	**C**	**D**
TC1a	*	1	1	1
TC1b	1	1	*	1
TC2	1	2	3	3
TC3a	*	3	1	4
TC3b	1	3	*	4
TC4	1	2	2	2
TC5	2	1	1	2
TC6a	*	2	2	4
TC6b	2	2	*	4
TC7	2	3	1	3
TC8	2	2	3	1
TC9	3	1	3	4
TC10	3	2	1	2
TC11a	*	3	2	1
TC11b	3	3	*	1
TC12a	*	1	2	3
TC12b	3	1	*	3
TC13	1	3	3	2

Step 2 - Replace				
	Parameters			
Test Case	**A**	**B**	**C**	**D**
TC1a	**2**	1	1	1
TC1b	1	1	**3**	1
TC2	1	2	3	3
TC3a	**3**	3	1	4
TC3b	1	3	**2**	4
TC4	1	2	2	2
TC5	2	1	1	2
TC6a	**1**	2	2	4
TC6b	2	2	**1**	4
TC7	2	3	1	3
TC8	2	2	3	1
TC9	3	1	3	4
TC10	3	2	1	2
TC11a	**1**	3	2	1
TC11b	3	3	**3**	1
TC12a	**1**	1	2	3
TC12b	3	1	**1**	3
TC13	1	3	3	2

Table 2: The two steps in the creation of a test suite satisfying pair-wise coverage for all valid sub-combinations of the hotel guide system test problem.

```
Let UC be a set of all pairs of values of any two parameters that are
not yet covered (initially all possible pairs).
Let TS be empty

While (test suite not empty)
      Get next test case from test suite
      if any pair in test case still exists in UC.
         Add test case to TS.
         Remove all test case pairs from UC.
      else
         Drop test case
      endif
      return TS
```

Figure 7: Simple algorithm for reducing the size of a test suite while maintaining pair-wise coverage.

2.5 Reducing the Test Suite

For reasons explained in the previous subsections, three of the four investigated constraint handling methods, abstract parameters, sub-models, and replace, may result in unnecessarily large test suites. Thus, we decided to include in this experiment a simple mechanism to reduce the size of the test suite. The test suite is given as input to the reducer. It processes the test cases one by one and any test case not adding new coverage is discarded, producing a possibly smaller test suite with the same level of coverage as the original test suite. Figure 7 shows an instance of the algorithm of the reducer adopted for pair-wise coverage.

3 Experiment Description

In this experiment four constraint handling methods are investigated. Three of these are investigated in both regular and reduced versions. Hence, in total, seven methods are investigated. The general framework of the experiment is to generate a test suite with a defined coverage from an input parameter model. Conflicts are added to the input parameter model and the seven constraint handling methods are used to generate conflict-free test suites.

The size of the test suite, i.e., the number of test cases, is used as the response variable to compare the different constraint handling methods. All in all, 2568 test suites with a total of $634,263$ test cases have been generated within the scope of this experiment. Figure 8 shows an

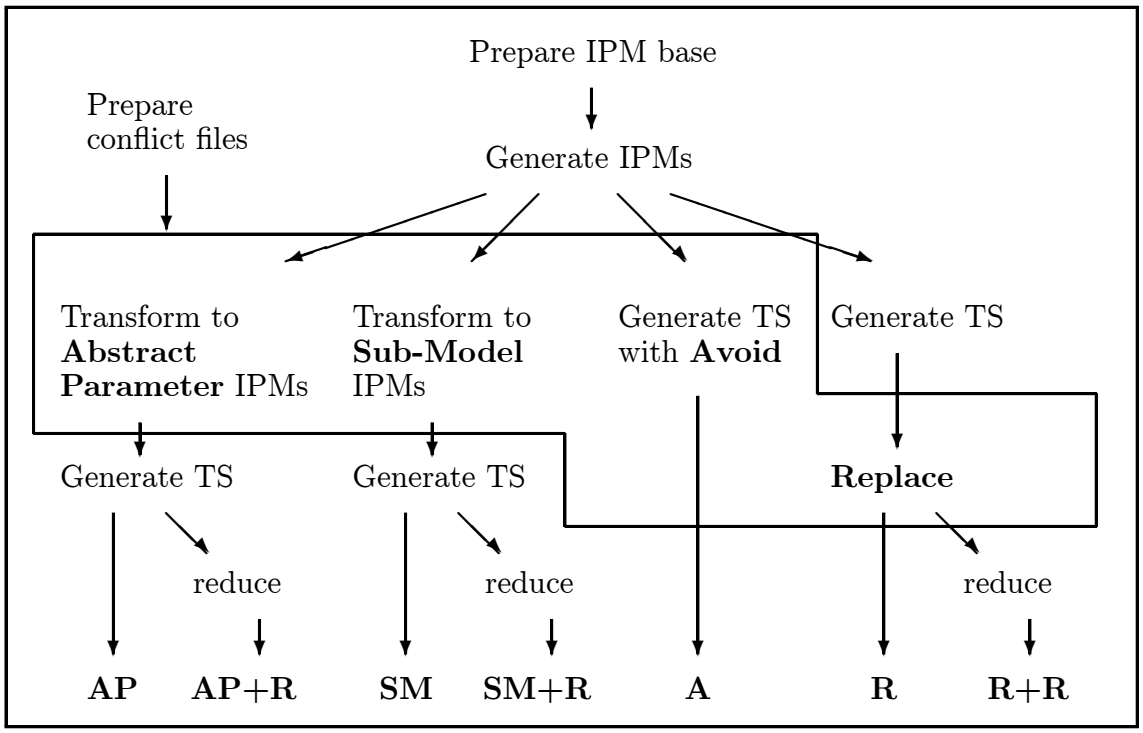

Figure 8: Experiment framework, which describes where the conflicts are handled by the seven constraint handling methods.

overview of how the constraint handling methods fit in the frame-work.

The execution of this experiment starts with preparation of the input parameter model (IPM) base and the conflict files. The IPM base contains a description of the sizes of the IPMs that should be included in the experiment. Each conflict file contains the sub-combinations of parameter values that should not be allowed when that conflict file is used. The preparations of the IPM and the conflict files are manual. The next step of this experiment is to automatically generate IPM files for the different sizes described in the IPM base. The same IPM files are then used by each of the four constraint handling methods. At this point the execution forks into four parallel tracks, one for each constraint handling method. In the two first methods each original IPM file is transformed accordingly. These tasks are fully automated for the abstract parameter method and semi-automated for the sub-model method. The next step in each of the four tracks is then to generate the test suite, which is fully automated. The same generation mechanism is used in all four tracks with the addition of a conflict avoidance mechanism for the avoid method. At this point all test suites are conflict-free except in the fourth track. In this case an automated replace mechanism is invoked removing the conflicts. Finally the same, automated, reduction mechanism is executed for the three tracks in which it may have an effect. All programs used in this experiment are implemented in Perl.

Controlled variables in this experiment are; *size* (with respect to number of parameters and

values) of the input parameter model, *coverage criterion* used, and *number* and *type* of conflicts. Table 3 shows an overview of the values of the controlled variables in this experiment and the following subsections describe and motivate these choices. All of the 432 combinations of these variables are investigated in this experiment.

IPM size			**Coverage criterion**	**Type of conflict**
# params	**# vars**	**balanced (Y/N)**		
5	4	Yes	1-wise	one pair
7	5	No	2-wise	two dependent pairs
9	6		3-wise	two independent pairs
11	7			
	8			
	9			

Table 3: Values of the controlled variables used in this experiment

3.1 Input Parameter Models

The input parameter model represents the input space of the test problem. The best way to create an input parameter model is still an open question. One approach is to use the actual parameters of the system under test and for each parameter select some interesting values. Another approach is to form abstract parameters based on the functionality of the system under test. Regardless of the approach, the end result is a set of parameters, each with some values. The input parameter model is used as input to the combination strategy to achieve some desired coverage 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 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. The time consumption of this experiment is the main determining factor for the choices of numbers of parameters and values. AETG, which is one of the combination strategies used in this experiment and described in the next subsection, has a time complexity of $O(V^4P^2log(P))$ where P is the number of parameters and V is the number of values of the largest parameter [TL02]. The abstract parameter constraint handling method is particularly time consuming since the constructed abstract parameter may contain as many values as the product of the number of values of the original parameters. Thus, limiting the number of values to a maximum of nine still produces

an abstract parameter with 80 values in the worst case. The $81st$ pair is invalid and thus not included.

For the number of values of each parameter there are two different situations. The IPM can either be unbalanced or balanced. In an unbalanced IPM different parameters may have different number of values whereas in balanced IPM all parameters contain the same number of values. Unbalanced IPMs are more natural. Using balanced IPMs gives results that are easier to analyze since there is one factor less that may influence the results. Taking these arguments into consideration we decided to use both.

The interpretation of an unbalanced IPM with P parameters and V values is that at least one parameter has V values and the rest of the parameters have anywhere 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. Again the reason was time consumption of the experiment execution. To our knowledge, the distribution of parameters across the parameters of IPMs in practice is not known; hence we opted for randomly assigned sizes to at least get some diversity.

Another influencing factor when selecting the sizes of the IPMs for this experiment is earlier research on combination strategies in which similar sizes have been used [CGMC03, GLOA06, STK04].

To summarize, this experiment is conducted with 48 different IPMs ranging in size from an IPM with five parameters with at the most four values each up to an IPM with eleven parameters each with nine values.

3.2 Implementations of Coverage Criteria

Whether or not there are conflicts in the input parameter model, the used coverage criterion greatly affects the number of test cases in the final test suite [GOA05]. As was explained in section 2.1, several of the investigated constraint handling methods need to be tailored for a specific coverage criterion. Thus, it is natural to include several different coverage criteria in this example.

The three coverage criteria used in this experiment are 1-wise, 2-wise, and 3-wise coverage, which require every parameter value, every pair of parameter values, and every triple of parameter values respectively, to be included in the final test suite. Time consumption is the main reason why these coverage criteria were used in this experiment. One aspect of time consumption is that these coverage criteria, in particular 1-wise and 2-wise coverage, require relatively few test cases. Another aspect of the time consumption is automation, which is a requirement for an experiment of this size. To generate test suite with the selected coverage, we decided to use the two combination strategies Each Choice (EC) (1-wise) and the algorithm from the Automatic Efficient Test Generator (AETG) (2-wise and 3-wise). These are both well suited for automation. Further, it is straightforward to add all of the investigated constraint handling methods to these combination strategies. Additional benefits from an experimental point of view with these combination strategies are that they are

```
Assume test cases t_1 − t_{i−1} already selected
Let UC be a set of all pairs of values of any two parameters that are
not yet covered by the test cases t_1 − t_{i−1}

A) Select candidates for t_i by
     1) Selecting the variable and the value included in most
        pairs in UC.
     2) Put the rest of the variables into a random order.
     3) For each variable in the sequence determined by step two,
        select the value included in most pairs in UC.
B) Repeat steps 1-3 k times and let t_i be the test case that covers
   the most pairs in UC.  Remove those pairs from UC.

Repeat until UC is empty.
```

Figure 9: An instance of the AETG algorithm for achieving pair-wise coverage.

well known and often used in previous research. The following subsections contain implementation details of these combination strategies.

3.2.1 Each Choice (EC)

The idea behind the *Each Choice* combination strategy, which was invented by Ammann and Offutt [AO94], is to include each value of each parameter in at least one test case. In its basic form test cases are successively selected by taking the next unused value of each parameter. If all values of a parameter are already covered, then as a general rule the first value of that parameter is selected. Integration of the avoid method for handling constraints into the each choice combination strategy results in a small deviation from this general rule. Instead of using the next unused value, the next unused value that will not cause a conflict is used.

3.2.2 Automatic Efficient Test Generator Algorithm (AETG)

Cohen et al. [CDPP96, CDFP97] described a greedy algorithm for achieving t-wise coverage, where t is an arbitrary number. This algorithm was implemented in the AETG tool. An instance for 2-wise coverage is presented in figure 9.

In this experiment, instances of the AETG algorithm are used to achieve both 2-wise and 3-wise coverage. The implementation for 2-wise coverage follows the algorithm in figure 9 with

some small additions. Steps 1, 3 and B may result in ties if more than one variable or test case have the same score. In our implementation all such cases are broken by random choices.

The implementation for 3-wise coverage is greatly inspired by the 2-wise implementation. The two main differences are that UC contains triples instead of pairs and that step 1 selects the pair included in most triples in UC.

Integration of the avoid constraint handling strategy resulted in checks each time a new parameter value is selected. These checks prevent the algorithm from selecting a parameter value in conflict with the already selected parameter values.

3.3 Conflicts

The selection of conflict types used in this experiment is based on our assumption that in practice it is more likely for conflicts in an input parameter models to be few and small rather than many and large. Thus, we focused our attention on conflicts involving one or two values each of two or three parameters.

We identified three types of conflicts involving two parameters; "one pair", "two dependent pairs", and "two independent pairs". In the one pair conflict, one value from each of two parameters, e.g., (A1, B1), conflict with each other. In the two dependent pairs conflict, one value of one parameter conflicts with two values of a second parameter. In our experimental set-up this is represented by two pairs, e.g., (A1, B1) and (A1, B2), hence the name. Finally, the two independent pairs conflict consists of two one pair conflicts of the same two parameters, e.g., (A1, B1) and (A2, B2).

The interpretation of a three-parameter conflict is not clear. Consider a three-parameter conflict involving one value of each parameter, e.g., (A1, B1, C1). There are at least two interpretations of this conflict. Either it is only the triple (A1, B1, C1) that is not allowed while all the possible pairs, e.g., (A1, B1) is allowed, or the triple (A1, B1, C1), as well as all the pairs are not allowed in the test suite. In the case of the latter interpretation the alternative representation of that conflict is $\{(A1,\ B1), (A1,\ C1), (B1,\ C1)\}$. Both for time consumption reasons and for the unclear interpretation we decided to leave the three-parameter conflicts for future study.

In four of the 24 cases with unbalanced IPMs, parameter B has only two values. Applying the two dependent pairs conflict ((A1, B1) and (A1, B2) are not allowed) makes it impossible to find a solution. Hence, these combinations were removed from the results.

3.4 Threats to Validity

The internal validity of an experiment concerns factors in the experiment set-up and execution that may distort the results. External validity is concerned with to what extent the results can be generalized.

It is a potential threat to the internal validity that the same persons both propose and then evaluate some of the methods. Intentional or unintentional bias may be introduced to make the

own methods appear better. Of the three methods that perform the best in this experiment two are proposed by the authors. Deliberate measures have been taken not to favor the own methods. For instance, the conflicts used in this experiment actually favor the third method, not proposed by the authors. Another example is the reduction algorithm, which is part of two of the suggested methods. It processes the test cases one by one as they appear in the test suite. It is likely that more elaborate reduction schemes will result in smaller test suites.

Another potential threat to the internal validity is the risk of faults in the scripts implementing the constraint handling methods and the combination strategies. As a consequence the implementations have been tested. Each script was executed on a few of the smaller IPMs and the results checked by hand to make sure that all valid values are still there and no invalid values are included in the results. Some checks for out-of-bounds values have also been implemented in the scripts. Several of the scripts are also driven by coverage and will not terminate unless 100% coverage is reached. Taken together, we believe that these measures give reasonable confidence in the correctness of the scripts.

A more severe threat to the internal validity is the use of random tie breaking in the implementations of the AETG method. The AETG combination strategy is a heuristic so to minimize the risk of unfavorable decisions, in each iteration 50 test cases candidates are generated and evaluated. This choice is based on the advice from Cohen et al. [CDPP96] that more than 50 test case candidates in each iteration will not significantly improve the final solution. Despite this, some test suites contain more or fewer test cases than expected, which can be observed by comparing the test suites of the base case and the avoid. The difference in test cases between these two methods should always be small. One way to compensate for this would be to remove results that appear to deviate from the expected. However, as this approach relies on judgment there is a risk that some true results would be removed. Another way of handling the problem with decisions based on randomness is to have a sufficiently large sample size. This is the approach used in this experiment. The comparison of the methods are based on 140 samples for each method.

A large problem when it comes to external validity is representativity [LGO04]. In the context of this experiment, neither the types nor sizes of conflicts in input parameter models nor the actual sizes of the IPMs in practice have been extensively studied. Thus, it is difficult to assess the representativity of results of this study. This is possibly the largest threat to validity of this experiment. Without the knowledge of what the reality looks like, the best we can do is to present our assumptions and to be careful in making general claims.

The last threat to external validity is lack of background information. The sub-model constraint handling method is sketched in a couple of background papers [WP96, CDFP97]. None of these papers contain a full description of 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 and described in this paper there is at least one alternative way to implement the algorithm. The general approach in such matters, which we used, is to clearly present the selected interpretation.

4 Results

The response variable in this experiment is the number of test cases in the test suites when using the investigated constraint handling methods. The number of test cases in the final test suite is certainly a key issue for the tester but there may also be other properties such as time consumption and ease of use that could influence the choice of method. After presenting the test case based results in the next subsection, a second subsection will cover other important aspects of the constraint handling methods.

4.1 Test Suite Sizes

Figures 10, 11, and 12 show box-plots of the number of test cases in the resulting test suites for the seven constraint handling methods with 1-wise, 2-wise, and 3-wise coverage respectively. A box-plot gives an overview of the distribution of values in a set. 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 *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 represent sizes of 140 test suites. These test suites are the results from taking the same 48 IPMs augmented with three different conflict types. Four IPMs are removed for one of the conflict types since there are no solutions to these problems. See section 3.3 for the details.

The order of the box-plots in each figure are; *B*: base, i.e., a reference with no conflicts, *AP*: abstract parameter, *AP+R*: reduced abstract parameter, *SM*: sub-models, *SM+R*: reduced sub-models, *A*: avoid, *R*: replace, and *R+R*: reduced replace. The descriptions of these methods are presented in section 2.

For 1-wise coverage, shown in figure 10, the results from AP+R are omitted since reduction does not yield any further improvements.

1-wise coverage of a conflict-free IPM results in a test suite with the same number of test cases as there are values of the largest parameter. In this experiment the number of values of the largest parameter range from 4 to 9, which is clearly visible in the base case.

All the methods except the sub-models method behave similarly. In each case, the abstract parameter method results in a new IPM with an abstract parameter with one or two values more, depending on the type of conflict, than in the original IPM.

The avoid method behaves exactly as the base in all 140 cases. The explanation is that the conflicts in this experiment are all encountered early during the execution when there are still plenty of uncovered values of each parameter to choose from. Only a conflict found in the last iteration of the avoid algorithm will result in one extra test case compared to the base. Thus, in this part of the experiment the avoid method is slightly favored.

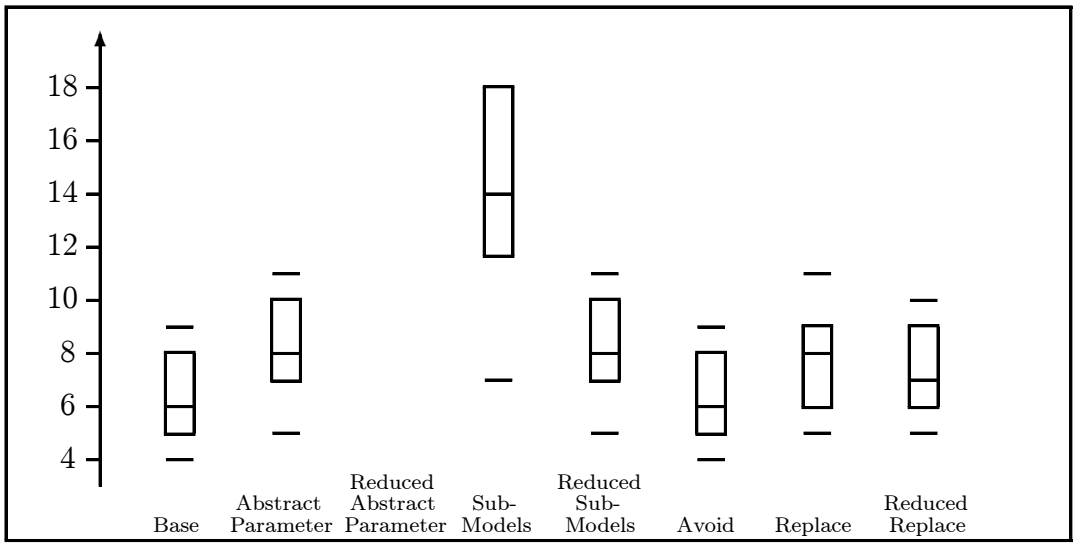

Figure 10: Box-plots of the sizes of test suites satisfying 1-wise coverage generated by the seven constraint handling methods. AP+R is omitted since reduction does not yield any further improvements. The upper adjacent value (27) for SM is omitted for space reasons

The replace method adds one test case to the test suite for each conflicting test case. In a test suite satisfying one-wise coverage each parameter value is included at least once. Thus, it may be the case that the conflicting values of two parameters end up in different test cases with a conflict-free test suite as a result. The implementation of Each Choice and the conflicts used in this experiment results in all 1-wise test suites having one or two conflicting test cases. This explains why the results for the replace method is skewed 1-2 units compared to the base case. However, it is definitely the case that the replace method is negatively biased in this part of the experiment. This is somewhat helped by reduction of the test suite which manages to remove one test case from about one third of the test suites.

Finally, the reason for the different behavior of the sub-models method is that the original IPM is divided into two or three sub-models, depending on the type of conflict. Each sub-model contains almost as many parameter values as the original IPM since the conflicts only affects two parameters. The result is that the number of test cases is nearly doubled for the two sub-model conflict types or nearly tripled for the conflict types that require three sub-models. Reducing the sub-models test suite removes most of the redundancy introduced by the sub-models.

None of the cases, have any outliers. The interpretation of this is that the test suites are relatively similar in size regardless of IPMs and conflicts.

In this experiment the test cases for 2-wise coverage are generated randomly so the positions of the conflicts will not affect the results as in the case with 1-wise coverage. The results for the 2-wise coverage is shown in figure 11.

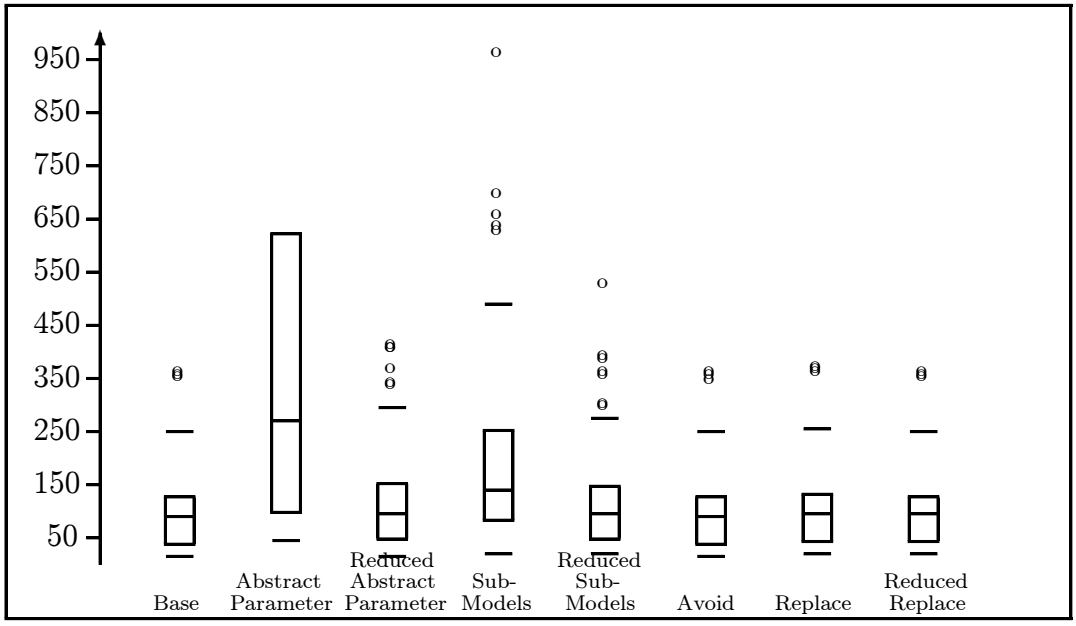

Figure 11: Box-plots of the sizes of test suites satisfying 2-wise coverage generated by the seven constraint handling methods. The upper adjacent value (1264) and the ten outliers (1413-3280) for AP are omitted for space reasons

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 test suite be reduced.

The test suite sizes for the base case range from 17 to 352 test cases. As in the case with 1-wise coverage the avoid and replace methods behave very similar to the base case. Both Chi-square and F-tests show with high probabilities that the hypotheses cannot be rejected that the three pairs base case and avoid, base case and replace, and base case and reduced replace are the same. Reduced avoid and reduced replace both yield test suites which are one or two test cases smaller than the corresponding base case test suite just as expected.

The abstract parameter method performs poorly, which is expected. For 2-wise coverage the abstract parameter has to contain all valid pairs of the two parameters involved in the conflict. Thus the size of the abstract parameter is close to the product of the sizes of the original parameters. This will greatly affect the size of the final test suite since it has to contain at least as many test cases as the product of the two largest parameters. Reduction of the test suites has a significant effect on the sized of the test suites. For the smaller test suites the behavior of the AP+R is similar to the base case. For instance, the lower adjacent values and the medians are

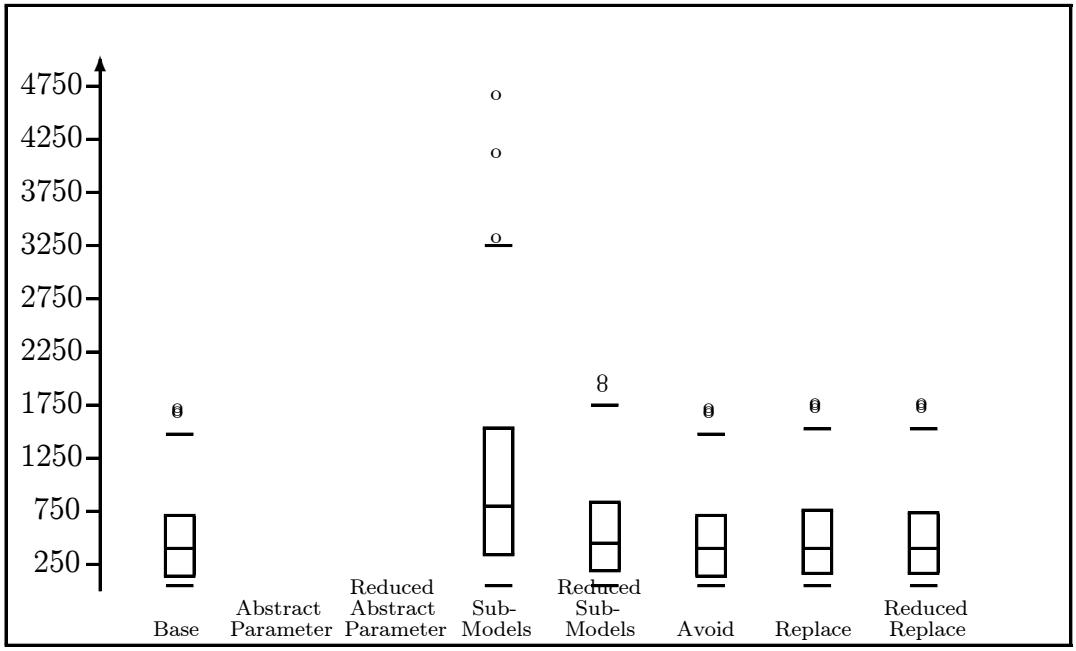

Figure 12: Box-plots of the sizes of test suites satisfying 3-wise coverage generated by the seven constraint handling methods. AP and AP+R are omitted due to time constraints.

similar. With larger test suites a difference is observable. Both the upper adjacent values and the outliers AP+R generates around 50 test cases more than in the corresponding base cases. AP+R also has six outliers compared to three for the base case. Both Chi-square and F-tests reject the hypothesis that the base case and AP+R are the same.

The sub-model method also performs worse than the base case. The explanation is the same as for 1-wise coverage, i.e., each sub-model, nearly as large as the original model, results in a test suite. The final test suite is the union of these sub-model test suites. Reduction of the final test suite yields results closer to the base case. Just as in the abstract parameter method the results of the smaller IPMs are more similar to the base case but with larger IPMs differences start to show. Again both Chi-square and F-tests reject the hypothesis that the base case and SM+R are the same.

Just as with 2-wise coverage, test cases for 3-wise coverage are generated randomly in this experiment so the positions of the conflicts will not affect the results as in the case with 1-wise coverage. The results for the 3-wise coverage is shown in figure 12.

In a conflict-free test suite with 3-wise coverage every triple of values of any three parameters are included in the test suite. This means that a test suite always must contain at least as many test cases as the product of the number of values of the three largest parameters. A conflict between two parameter values means that any triple including this pair of values should not be included in the test suite.

The results from 3-wise coverage have much in common with the results from 2-wise coverage. Just as in the earlier cases the avoid, replace, and reduced replace methods perform similar to the base case. With high probability, both a chi-square test and an F-test cannot reject the hypothesis that the base case and avoid are the same. However, for replace and reduced replace the results are different. With around 83% probability the F-test cannot reject the hypothesis that the base case and replace, and the base case and reduced replace respectively are the same but the chi-square test rejects both these hypotheses. Although difficult to see in figure 12, it is the larger IPMs that are the cause for this. More parameters or more values for each parameter result in more triples containing the conflicting pair. For the avoid method this means fewer triples to cover which may even require fewer test cases than the base case, whereas the replace method will add one new test case for each test case a conflict. Thus it is not surprising that the replace method is outperformed. What is somewhat surprising is that reduction does not yield a better effect.

As in the previous cases the sub-models method performs worse than the base case. The same explanation, with the sub-models nearly of the same size as the original IPM, still applies. Also as in the previous cases, reduction has a significant effect on the size of the test suite but never reaches the level of the base case.

The abstract parameter method was omitted due to time constraints. The same abstract parameter as in 2-wise coverage can be used but its size has to be multiplied with the sizes of two more parameters to get a minimal size of the final test suite. With the achieved result for 1-wise and 2-wise coverage, it was deemed too time consuming to proceed with 3-wise coverage. Without the test suites from the abstract parameter method it was not possible to achieve any reduced results.

4.2 Time Consumption, Ease of Use and Applicability

In addition to the size of the test suite other properties of the constraint handling methods may also affect the choice of the tester. Three such properties are time consumption, ease of use and applicability of the constraint handling methods.

No time consumption metrics were collected during this experiment. Despite this, some important observations were made. All four methods (abstract parameter, sub-models, avoid, and replace) require some implementation. Both the avoid and the replace methods are easy to fully automate although the avoid method require one implementation for each combination strategy. The abstract parameter and sub-model methods are more difficult to automate since these require transformations of the IPM.

In the case of the abstract parameter two transformation steps are required, first the conflicts of the original IPM are removed through a transformation into an IPM with abstract parameters. Then, the test suite is generated which contains values of the abstract parameter. Finally, the test suite has to be transformed back to the original parameters. In the general case, the sub-model transformations may be recursive with different steps in different recursions. In this experiment scripts were developed to do some of these transformation steps but some manual intervention was

also used.

In the execution phase, i.e., when the test suites are generated, the time consumption is dominated by the test case generation not the constraint handling. For instance, the test case generation for the base case took several days while using the replace method on the generated test suites took less than one hour. The abstract parameter method was the only method that introduced any noticeable overhead. The reason for this is that the two of the parameters of the original IPM is transformed into one new parameter with many more values. In the combination strategies the maximum number of values of any parameter has a profound effect on the time consumption.

From a usability perspective the avoid method is the simplest since it is completely integrated in the combination strategy algorithm. All the user has to do is to prepare the conflict description and then start the test case generation. The replace method is almost as simple since the only additional thing required by the user is to invoke the implementation of the replacement method. Thus, the user does not need any detailed understanding of these two methods to use them. Depending on the level of automation the tester may need more or less knowledge of the abstract parameter and sub-model methods to use them.

A final observation concerns the applicability of the four methods. The abstract parameter, sub-models, and replace methods are all completely general with respect to the combination strategy used. The reason is that conflicts are handled either before or after the combination strategy is applied. The avoid method has to be integrated into the combination strategy. For some combination strategies this may not be possible. The orthogonal method [WP96] is one example of a combination strategy which cannot be used with the avoid method.

5 Conclusions and Future Research

Table 4 contains an attempt to classify the results in this experiment. The levels of the different categories were selected to highlight the major differences between the methods, thus some details are hidden and others may be a bit magnified.

In this experiment the avoid, replace, and reduced replace methods stand out as the best constraint handling methods. Not only do they produce small test suites, they are also easy to use and once implemented, cheap in terms of time consumption. Cohen et al. [CDFP97] 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. This experiment generally agree these observations although the differences are smaller than one order of magnitude.

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.

	Sub-Models		Abstract Parameter		Avoid	Replace	
Category	**regular**	**reduced**	**regular**	**reduced**	**regular**	**regular**	**reduced**
Small test suites	No	Yes	No	Yes	Yes	Yes	Yes
Little preparations	No	No	No	No	Yes	Yes	Yes
Short execution time	Yes	Yes	No	No	Yes	Yes	Yes
Easy to use	No	No	No	No	Yes	Yes	Yes
Generally applicable	Yes	Yes	Yes	Yes	No	Yes	Yes

Table 4: Compact summary of the results and observations of this experiment.

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 invalid sub-combinations, these can be resolved automatically and efficiently at a later stage. Our interpretation of the results from this experiment is that it is preferable to keep the number of IPMs, parameters and values low by allowing conflicts than to have more and larger IPMs without conflicts.

Whether to use the avoid, the replace or the reduced replace methods is impossible to say based on the results of this experiment. In most cases the avoid method results in fewer test cases that the two others but the differences are small and not statistically valid. 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 should not 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 constraint handling method decreases with higher coverage. When using N-wise coverage, there is only a single set of combinations that satisfies this goal, every valid 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 are also interesting to investigate.

6 Bibliography

References

[AO94] Paul E. Ammann and A. Jefferson Offutt. Using formal methods to derive test frames in category-partition testing. In *Proceedings of the Ninth Annual Conference on Computer Assurance (COMPASS'94),Gaithersburg MD*, pages 69–80. IEEE Computer Society Press, June 1994.

[CDFP97] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The AETG System: An Approach to Testing Based on Combinatorial Design. *IEEE Transactions on Software Engineering*, 23(7):437–444, July 1997.

[CDPP96] David M. Cohen, Siddhartha R. Dalal, Jesse Parelius, and Gardner C. Patton. The Combinatorial Design Approach to Automatic Test Generation. *IEEE Software*, 13(5):83–89, September 1996.

[CGMC03] M.B. Cohen, P.B. Gibbons, W.B. Mugridge, and C.J. 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.

[DHS02] Nigel Daley, Daniel Hoffman, and Paul Strooper. A framework for table driven testing of Java classes. *Software - Practice and Experience (SP&E)*, 32(5):465–493, April 2002.

[GLOA06] Mats Grindal, Birgitta Lindström, A. Jefferson Offutt, and Sten F. Andler. An evaluation of combination strategies for test case selection. *Empirical Software Engineering*, To appear, 2006.

[GOA05] M. Grindal, J. Offutt, and S.F. Andler. Combination testing strategies: A survey. *Software Testing, Verification, and Reliability*, 15(3):167–199, September 2005.

[LGO04] Birgitta Lindström, Mats Grindal, and A. Jefferson 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), November 2004.

[Man85] Robert Mandl. Orthogonal Latin Squares: An application of experiment design to compiler testing. *Communications of the ACM*, 28(10):1054–1058, October 1985.

[OB88] Thomas J. Ostrand and Marc J. Balcer. The Category-Partition Method for Specifying and Generating Functional Tests. *Communications of the ACM*, 31(6):676–686, June 1988.

[STK04] Toshiaki Shiba, Tatsuhiro Tsuchiya, and Tohru Kikuno. Using artificial life techniques to generate test cases for combinatorial testing. In *Proceedings of 28th Annual Interna-*

tional Computer Software and Applications Conference (COMPSAC'04) 2004, Hong Kong, China, 28-30 September 2004, pages 72–77. IEEE Computer Society, 2004.

[TL02] Kuo-Chung Tai and Yu Lei. A Test Generation Strategy for Pairwise Testing. *IEEE Transactions on Software Engineering*, 28(1):109–111, January 2002.

[WP96] Alan W. Williams and Robert L. Probert. A practical strategy for testing pair-wise coverage of network interfaces. In *Proceedings of the 7th International Symposium on Software Reliability Engineering (ISSRE 96), White Plains, New York, USA, Oct 30 - Nov 2, 1996*, pages 246–254, Nov 1996.

[WP01] Alan W. Williams and Robert L. Probert. A measure for component interaction test coverage. In *Proceedings of the ACSI/IEEE International Conference on Computer Systems and Applications (AICCSA 2001), Beirut, Lebanon, June 2001*, pages 304–311, June 2001.

A Input Parameter Models

This appendix describes the contents of the 48 input parameter models used in this experiment.

P5V4V	(4,4,2,3,2)	P9V4V	(4,3,4,3,4,4,3,2,3)
P5V5V	(5,3,5,2,2)	P9V5V	(5,2,4,2,2,5,4,4,2)
P5V6V	(6,6,5,6,2)	P9V6V	(6,4,4,4,5,2,3,2,3)
P5V7V	(7,7,7,2,4)	P9V7V	(7,2,5,5,7,5,3,5,7)
P5V8V	(8,8,8,6,7)	P9V8V	(8,2,7,5,3,7,7,3,8)
P5V9V	(9,3,9,3,9)	P9V9V	(9,7,4,8,5,7,5,8,9)
P7V4V	(4,3,2,4,4,4,2)	P11V4V	(4,4,3,3,2,2,4,4,3,3,3)
P7V5V	(5,3,4,4,5,3,5)	P11V5V	(5,4,3,3,3,5,5,3,4,3,5)
P7V6V	(6,3,2,6,4,2,3)	P11V6V	(6,6,3,4,3,6,2,4,4,6,3)
P7V7V	(7,2,7,4,3,6,3)	P11V7V	(7,6,4,6,6,3,4,4,6,6,2)
P7V8V	(8,5,3,5,8,6,5)	P11V8V	(8,4,8,7,6,7,4,5,6,3,3)
P7V9V	(9,3,8,3,9,2,7)	P11V9V	(9,3,7,3,9,3,9,3,7,7,5)
P5V4C	(4,4,4,4,4)	P9V4C	(4,4,4,4,4,4,4,4,4)
P5V5C	(5,5,5,5,5)	P9V5C	(5,5,5,5,5,5,5,5,5)
P5V6C	(6,6,6,6,6)	P9V6C	(6,6,6,6,6,6,6,6,6)
P5V7C	(7,7,7,7,7)	P9V7C	(7,7,7,7,7,7,7,7,7)
P5V8C	(8,8,8,8,8)	P9V8C	(8,8,8,8,8,8,8,8,8)
P5V9C	(9,9,9,9,9)	P9V9C	(9,9,9,9,9,9,9,9,9)
P7V4C	(4,4,4,4,4,4,4)	P11V4C	(4,4,4,4,4,4,4,4,4,4,4)
P7V5C	(5,5,5,5,5,5,5)	P11V5C	(5,5,5,5,5,5,5,5,5,5,5)
P7V6C	(6,6,6,6,6,6,6)	P11V6C	(6,6,6,6,6,6,6,6,6,6,6)
P7V7C	(7,7,7,7,7,7,7)	P11V7C	(7,7,7,7,7,7,7,7,7,7,7)
P7V8C	(8,8,8,8,8,8,8)	P11V8C	(8,8,8,8,8,8,8,8,8,8,8)
P7V9C	(9,9,9,9,9,9,9)	P11V9C	(9,9,9,9,9,9,9,9,9,9,9)

Table 5: Number of values of each parameter in the 24 variable and 24 complete IPMs.

B Results

The following tables contain the sizes of the generated test suites in this example.

	1-wise		**2-wise**		**3 -wise**	
	V	C	V	C	V	C
P5V4	4	4	17	23	52	93
P5V5	5	5	28	38	76	179
P5V6	6	6	42	84	233	304
P5V7	7	7	58	78	350	469
P5V8	8	8	89	103	593	701
P5V9	9	9	93	130	735	987
P7V4	4	4	24	32	89	117
P7V5	5	5	34	50	155	225
P7V6	6	6	41	72	155	379
P7V7	7	7	54	104	316	588
P7V8	8	8	76	143	435	866
P7V9	9	9	101	181	702	1214
P9V4	4	4	27	35	101	141
P9V5	5	5	32	57	136	268
P9V6	6	6	38	88	159	461
P9V7	7	7	76	125	416	717
P9V8	8	8	94	175	612	1055
P9V9	9	9	138	243	895	1486
P11V4	4	4	32	49	99	156
P11V5	5	5	50	84	188	302
P11V6	6	6	69	127	310	507
P11V7	7	7	90	186	412	792
P11V8	8	8	128	251	622	1178
P11V9	9	9	165	352	901	1654

Table 6: Number of test cases in each test suite generated from the 24 (V)ariable and 24 (C)omplete conflict-free IPMs for the three levels of coverage.

	Abstract Parameter 1-wise					
	V			**C**		
	C1	C2	C3	C1	C2	C3
P5V4	5	6	6	5	6	6
P5V5	6	7	7	6	7	7
P5V6	7	8	8	7	8	8
P5V7	8	9	9	8	9	9
P5V8	9	10	10	9	10	10
P5V9	10	11	11	10	11	11
P7V4	5	6	6	5	6	6
P7V5	6	7	7	6	7	7
P7V6	7	8	8	7	8	8
P7V7	8	-	9	8	9	9
P7V8	9	10	10	9	10	10
P7V9	10	11	11	10	11	11
P9V4	5	6	6	5	6	6
P9V5	6	-	7	6	7	7
P9V6	7	8	8	7	8	8
P9V7	8	-	9	8	9	9
P9V8	9	-	10	9	10	10
P9V9	10	11	11	10	11	11
P11V4	5	6	6	5	6	6
P11V5	6	7	7	6	7	7
P11V6	7	8	8	7	8	8
P11V7	8	9	9	8	9	9
P11V8	9	10	10	9	10	10
P11V9	10	11	11	10	11	11

Table 7: Sizes of 1-wise covering test suites using abstract parameter to handle conflicts. Results from 24 (V)ariable and 24 (C)omplete IPMs with three different conflicts (C1, C2, and C3) are included.

	Sub-models 1-wise						Sub-models reduced 1-wise					
	V			**C**			**V**			**C**		
	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
P5V4	7	7	10	8	8	12	5	6	6	5	6	6
P5V5	10	10	15	10	10	15	6	7	7	6	7	7
P5V6	12	12	18	12	12	18	7	8	8	7	8	8
P5V7	14	14	21	14	14	21	8	9	9	8	9	9
P5V8	16	16	24	16	16	24	9	10	10	9	10	10
P5V9	18	18	27	18	18	27	10	11	11	10	11	11
P7V4	8	8	12	8	8	12	5	6	6	5	6	6
P7V5	10	10	15	10	10	15	6	7	7	6	7	7
P7V6	12	12	18	12	12	18	7	8	8	7	8	8
P7V7	14	-	14	14	14	21	8	-	8	8	9	9
P7V8	16	16	24	16	16	24	9	10	10	9	10	10
P7V9	18	18	27	18	18	27	10	11	11	10	11	11
P9V4	8	8	12	8	8	12	5	6	6	5	6	6
P9V5	10	-	10	10	10	15	6	-	6	6	7	7
P9V6	11	11	16	12	12	18	7	8	8	7	8	8
P9V7	14	-	14	14	14	21	8	-	8	8	9	9
P9V8	16	-	16	16	16	24	9	-	9	9	10	10
P9V9	18	18	27	18	18	27	10	11	11	10	11	11
P11V4	8	8	12	8	8	12	5	6	6	5	6	6
P11V5	10	10	15	10	10	15	6	7	7	6	7	7
P11V6	12	12	18	12	12	18	7	8	8	7	8	8
P11V7	13	13	19	14	14	21	8	9	9	8	9	9
P11V8	16	16	24	16	16	24	9	10	10	9	10	10
P11V9	18	18	27	18	18	27	10	11	11	10	11	11

Table 8: Sizes of 1-wise covering test suites using regular and reduced sub-IPMs to handle conflicts. Results from 24 (V)ariable and 24 (C)omplete IPMs with three different conflicts (C1, C2, and C3) are included.

	Avoid 1-wise					
	V			**C**		
	C1	C2	C3	C1	C2	C3
P5V4	4	4	4	4	4	4
P5V5	5	5	5	5	5	5
P5V6	6	6	6	6	6	6
P5V7	7	7	7	7	7	7
P5V8	8	8	8	8	8	8
P5V9	9	9	9	9	9	9
P7V4	4	4	4	4	4	4
P7V5	5	5	5	5	5	5
P7V6	6	6	6	6	6	6
P7V7	7	-	7	7	7	7
P7V8	8	8	8	8	8	8
P7V9	9	9	9	9	9	9
P9V4	4	4	4	4	4	4
P9V5	5	-	5	5	5	5
P9V6	6	6	6	6	6	6
P9V7	7	-	7	7	7	7
P9V8	8	-	8	8	8	8
P9V9	9	9	9	9	9	9
P11V4	4	4	4	4	4	4
P11V5	5	5	5	5	5	5
P11V6	6	6	6	6	6	6
P11V7	7	7	7	7	7	7
P11V8	8	8	8	8	8	8
P11V9	9	9	9	9	9	9

Table 9: Sizes of 1-wise covering test suites using avoid to handle conflicts. Results from 24 (V)ariable and 24 (C)omplete IPMs with three different conflicts (C1, C2, and C3) are included.

	Replace 1-wise						Replace reduced 1-wise					
	V			**C**			**V**			**C**		
	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
P5V4	5	5	6	5	5	6	5	5	5	5	5	5
P5V5	6	6	7	6	6	7	6	6	6	6	6	6
P5V6	7	7	8	7	7	8	7	7	7	7	7	7
P5V7	8	8	9	8	8	9	8	8	8	8	8	8
P5V8	9	9	10	9	9	10	9	9	9	9	9	9
P5V9	10	10	11	10	10	11	10	10	10	10	10	10
P7V4	5	5	6	5	5	6	5	5	5	5	5	5
P7V5	6	6	7	6	6	7	6	6	6	6	6	6
P7V6	7	7	8	7	7	8	7	7	7	7	7	7
P7V7	8	-	9	8	8	9	8	-	8	8	8	8
P7V8	9	9	10	9	9	10	9	9	9	9	9	9
P7V9	10	10	11	10	10	11	10	10	10	10	10	10
P9V4	5	5	6	5	5	6	5	5	5	5	5	5
P9V5	6	-	7	6	6	7	6	-	6	6	6	6
P9V6	7	7	8	7	7	8	6	6	7	7	7	7
P9V7	8	-	9	8	8	9	8	-	8	8	8	8
P9V8	9	-	10	9	9	10	9	-	9	9	9	9
P9V9	10	10	11	10	10	11	10	10	10	10	10	10
P11V4	5	5	6	5	5	6	5	5	5	5	5	5
P11V5	6	6	7	6	6	7	6	6	6	6	6	6
P11V6	7	7	8	7	7	8	7	7	7	7	7	7
P11V7	8	8	9	8	8	9	7	7	8	8	8	8
P11V8	9	9	10	9	9	10	9	9	9	9	9	9
P11V9	10	10	11	10	10	11	10	10	10	10	10	10

Table 10: Sizes of 1-wise covering test suites using regular and reduced replace to handle conflicts. Results from 24 (V)ariable and 24 (C)omplete IPMs with three different conflicts (C1, C2, and C3) are included.

	Abstract Parameter 2-wise						Abstract Parameter reduced 1-wise					
	V			**C**			**V**			**C**		
	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
P5V4	48	43	43	75	70	70	20	17	17	31	33	31
P5V5	70	66	66	156	150	150	29	28	26	54	56	53
P5V6	231	233	233	287	278	278	63	65	67	89	75	76
P5V7	365	350	350	469	459	459	86	76	76	118	110	113
P5V8	645	632	632	708	700	700	141	136	135	167	152	152
P5V9	272	255	255	1030	1034	1034	122	117	117	201	199	200
P7V4	48	44	44	76	68	68	30	30	30	37	36	38
P7V5	80	74	74	161	155	155	46	42	42	64	65	65
P7V6	102	98	98	304	284	284	44	41	41	100	92	94
P7V7	91	-	88	503	473	473	62	-	60	138	121	124
P7V8	341	328	328	747	736	736	101	94	95	188	187	186
P7V9	249	242	242	1134	1113	1113	123	115	115	256	233	233
P9V4	58	53	53	93	85	85	33	34	33	49	50	50
P9V5	55	-	51	198	190	190	40	-	41	83	80	81
P9V6	151	149	149	387	384	384	50	48	46	118	117	120
P9V7	128	-	118	723	740	740	95	-	92	170	161	162
P9V8	167	-	159	1550	1413	1413	121	-	123	245	241	243
P9V9	1561	1264	1264	3256	3280	3280	203	198	199	358	334	334
P11V4	73	68	68	94	80	80	31	32	33	54	52	51
P11V5	118	114	114	199	186	186	59	57	58	94	101	100
P11V6	267	259	259	359	357	357	81	79	80	152	152	151
P11V7	366	359	359	647	620	620	103	98	98	200	214	212
P11V8	314	315	315	975	934	934	139	139	140	284	296	296
P11V9	310	295	295	1555	1528	1528	181	179	180	399	404	405

Table 11: Sizes of 1-wise covering test suites using regular and reduced abstract parameter to handle conflicts. Results from 24 (V)ariable and 24 (C)omplete IPMs with three different conflicts (C1, C2, and C3) are included.

	Sub-models 2-wise						Sub-models reduced 2-wise					
	V			**C**			**V**			**C**		
	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
P5V4	25	22	33	41	43	59	18	19	26	27	32	35
P5V5	48	47	65	63	64	92	33	36	34	39	45	61
P5V6	72	73	100	96	91	139	46	53	70	59	61	93
P5V7	94	93	134	135	127	185	59	66	94	81	87	120
P5V8	149	141	207	176	174	254	92	95	145	106	110	165
P5V9	178	173	260	234	222	325	100	108	115	139	143	213
P7V4	45	45	63	57	57	81	28	35	32	34	44	49
P7V5	68	66	96	86	89	126	41	48	49	51	57	81
P7V6	66	67	99	136	131	194	42	50	49	82	90	122
P7V7	106	-	102	187	184	282	63	-	64	110	118	171
P7V8	138	142	210	248	248	374	82	93	114	139	153	230
P7V9	190	187	279	329	329	474	108	117	123	190	207	294
P9V4	49	46	73	70	66	95	30	35	40	41	48	57
P9V5	62	-	60	109	108	161	38	-	42	63	76	91
P9V6	69	70	98	171	171	249	41	50	57	99	109	149
P9V7	141	-	140	244	225	338	81	-	87	136	140	210
P9V8	187	-	184	329	320	479	103	-	112	181	192	294
P9V9	254	244	360	434	430	630	145	151	214	247	258	380
P11V4	54	52	77	89	84	134	35	42	51	54	62	72
P11V5	93	92	133	142	146	216	55	65	77	84	96	123
P11V6	119	112	173	233	229	340	71	77	111	128	142	191
P11V7	169	159	233	340	317	488	99	106	146	189	194	272
P11V8	240	252	359	482	477	689	134	154	160	268	273	384
P11V9	317	323	461	649	619	955	179	194	190	350	357	519

Table 12: Sizes of 2-wise covering test suites using regular and reduced sub-IPMs to handle conflicts. Results from 24 (V)ariable and 24 (C)omplete IPMs with three different conflicts (C1, C2, and C3) are included.

	Avoid 2-wise					
	V			**C**		
	C1	C2	C3	C1	C2	C3
P5V4	19	16	15	25	25	22
P5V5	27	28	26	35	35	37
P5V6	44	44	44	56	56	56
P5V7	53	55	58	76	76	78
P5V8	90	89	86	104	98	100
P5V9	94	93	93	133	132	129
P7V4	24	24	23	30	30	31
P7V5	35	36	34	48	49	48
P7V6	41	39	37	73	74	72
P7V7	57	-	57	103	102	103
P7V8	74	74	75	139	143	139
P7V9	100	99	94	182	177	180
P9V4	26	25	25	35	36	38
P9V5	32	-	33	62	60	57
P9V6	35	36	37	92	88	91
P9V7	74	-	76	126	124	127
P9V8	98	-	97	177	178	177
P9V9	136	133	130	238	239	230
P11V4	32	31	32	48	44	47
P11V5	54	51	50	80	77	75
P11V6	70	72	72	126	126	121
P11V7	88	92	87	175	177	176
P11V8	122	128	130	253	246	252
P11V9	162	159	160	354	342	349

Table 13: Sizes of 2-wise covering test suites using avoid to handle conflicts. Results from 24 (V)ariable and 24 (C)omplete IPMs with three different conflicts (C1, C2, and C3) are included.

	Replace 2-wise						Replace reduced 2-wise					
	V			**C**			**V**			**C**		
	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
P5V4	18	19	19	25	27	26	18	17	17	24	26	25
P5V5	29	32	31	39	41	40	28	28	29	38	40	38
P5V6	43	44	44	55	56	56	42	43	43	55	55	55
P5V7	60	61	63	81	82	82	59	59	61	80	81	80
P5V8	90	91	91	104	105	106	90	91	90	104	105	105
P5V9	95	101	100	132	133	133	95	99	99	132	133	131
P7V4	25	27	27	33	36	34	25	26	25	32	33	34
P7V5	36	37	37	52	54	53	36	37	37	52	53	53
P7V6	44	45	45	74	77	76	44	44	44	74	76	76
P7V7	60	-	63	108	109	110	57	-	59	104	106	106
P7V8	77	79	79	144	145	145	77	77	79	144	144	143
P7V9	104	106	110	182	183	184	103	104	104	181	182	183
P9V4	29	32	31	36	39	37	28	31	28	35	37	36
P9V5	36	-	40	59	61	60	36	-	38	59	61	59
P9V6	39	40	40	89	91	91	38	39	39	88	91	88
P9V7	81	-	82	132	133	135	80	-	81	125	126	128
P9V8	98	-	104	177	179	178	97	-	99	176	176	175
P9V9	140	141	141	250	254	255	140	141	141	246	250	246
P11V4	36	37	38	50	52	54	36	37	35	49	50	50
P11V5	53	54	57	85	86	87	53	54	53	82	82	83
P11V6	70	73	72	130	131	133	68	72	68	126	126	126
P11V7	93	96	96	189	191	193	90	92	91	184	185	185
P11V8	129	131	134	255	256	257	129	130	131	251	250	250
P11V9	174	178	179	362	365	364	167	167	166	350	351	350

Table 14: Sizes of 2-wise covering test suites using regular and reduced replace to handle conflicts. Results from 24 (V)ariable and 24 (C)omplete IPMs with three different conflicts (C1, C2, and C3) are included.

	Sub-models 3-wise						Sub-models reduced 3-wise					
	V			**C**			**V**			**C**		
	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
P5V4	59	50	67	151	153	216	49	48	47	102	120	118
P5V5	90	93	130	292	286	417	70	92	90	189	219	211
P5V6	352	344	492	507	508	725	233	259	254	320	353	351
P5V7	464	418	583	811	790	1163	338	332	331	497	538	535
P5V8	947	910	1321	1214	1193	1747	596	623	619	738	794	795
P5V9	1388	1384	2031	1725	1698	2486	824	901	903	1025	1099	1086
P7V4	152	154	225	206	209	301	102	128	125	134	165	157
P7V5	286	288	418	399	393	584	181	218	208	245	289	276
P7V6	285	283	411	674	677	985	186	215	213	407	468	451
P7V7	576	-	833	1069	1051	1566	361	-	356	629	699	686
P7V8	797	796	1174	1576	1564	2305	482	559	544	911	993	975
P7V9	1328	1319	1174	2219	2204	3245	776	874	851	1275	1391	1357
P9V4	181	178	263	257	258	380	120	146	141	159	202	184
P9V5	248	-	366	501	496	734	159	-	162	302	359	336
P9V6	274	278	405	854	845	1253	173	216	203	506	592	557
P9V7	792	-	1172	1326	1326	1953	475	-	479	771	882	843
P9V8	1167	-	1736	1962	1949	2882	681	-	687	1129	1261	1216
P9V9	1667	1639	2421	2273	2750	4063	979	1083	1045	1577	1720	1671
P11V4	169	171	245	291	287	430	115	139	137	180	227	208
P11V5	350	355	514	556	556	824	225	271	264	336	406	384
P11V6	502	478	706	954	955	1410	324	355	350	557	664	616
P11V7	731	722	1060	1502	1491	2216	449	523	496	861	997	934
P11V8	1194	1188	1749	2220	2207	3278	711	819	783	1254	1418	1346
P11V9	1713	1736	2544	3136	3114	4629	988	1151	1094	1754	1953	1865

Table 15: Sizes of 3-wise covering test suites using regular and reduced sub-IPMs to handle conflicts. Results from 24 (V)ariable and 24 (C)omplete IPMs with three different conflicts (C1, C2, and C3) are included.

	Avoid 3-wise					
	V			**C**		
	C1	C2	C3	C1	C2	C3
P5V4	49	48	44	92	87	90
P5V5	71	74	68	174	174	176
P5V6	230	229	255	299	301	296
P5V7	344	333	338	470	468	470
P5V8	582	587	574	688	695	695
P5V9	732	731	730	990	982	975
P7V4	90	85	88	116	115	117
P7V5	159	155	160	221	223	223
P7V6	154	153	149	378	376	376
P7V7	321	-	312	585	585	584
P7V8	432	443	427	846	846	859
P7V9	692	701	702	1218	1216	1213
P9V4	99	97	97	141	141	142
P9V5	135	-	135	273	273	270
P9V6	158	165	156	459	459	455
P9V7	422	-	415	723	717	712
P9V8	621	-	611	1054	1051	1059
P9V9	899	887	892	1481	1485	1487
P11V4	99	99	100	152	154	152
P11V5	189	196	192	299	298	301
P11V6	304	303	297	512	509	503
P11V7	413	409	406	800	794	800
P11V8	612	615	615	1169	1172	1175
P11V9	898	895	893	1654	1651	1651

Table 16: Sizes of 3-wise covering test suites using avoid to handle conflicts. Results from 24 (V)ariable and 24 (C)omplete IPMs with three different conflicts (C1, C2, and C3) are included.

	Replace 3-wise						Replace reduced 3-wise					
	V			**C**			**V**			**C**		
	C1	C2	C3	C1	C2	C3	C1	C2	C3	C1	C2	C3
P5V4	55	58	58	97	102	102	50	50	48	95	100	101
P5V5	81	86	86	185	193	192	74	73	70	182	188	184
P5V6	240	246	247	313	321	322	236	237	234	311	316	318
P5V7	357	365	364	477	489	486	345	342	342	475	485	483
P5V8	603	612	612	711	722	722	598	604	602	707	717	715
P5V9	759	792	791	999	1010	1013	735	735	735	994	1006	1004
P7V4	97	103	105	125	132	133	95	99	97	124	131	128
P7V5	163	173	173	234	243	243	163	171	167	233	242	240
P7V6	163	172	171	389	400	400	156	157	157	388	400	397
P7V7	340	-	359	599	610	609	323	-	328	598	609	607
P7V8	450	458	460	878	891	892	445	451	449	878	889	890
P7V9	728	754	753	1228	1241	1245	710	725	726	1228	1241	1242
P9V4	110	119	119	150	158	159	109	115	114	150	158	154
P9V5	150	-	160	277	286	288	143	-	149	276	285	287
P9V6	166	174	172	473	488	487	166	171	168	472	484	484
P9V7	449	-	479	732	748	746	443	-	467	731	746	742
P9V8	645	-	686	1068	1086	1086	635	-	667	1068	1085	1082
P9V9	908	920	921	1505	1521	1521	908	920	918	1501	1519	1515
P11V4	106	112	112	166	178	174	103	109	108	165	176	172
P11V5	197	204	206	315	328	326	197	204	205	314	326	324
P11V6	318	328	327	519	536	532	315	322	319	518	534	530
P11V7	423	431	433	809	827	823	421	427	429	807	825	821
P11V8	639	654	657	1196	1215	1213	639	653	654	1195	1214	1212
P11V9	939	975	975	1678	1698	1699	935	967	962	1678	1697	1697

Table 17: Sizes of 3-wise covering test suites using regular and reduced replace to handle conflicts. Results from 24 (V)ariable and 24 (C)omplete IPMs with three different conflicts (C1, C2, and C3) are included.