FORMULATION OF THE INTERACTION TEST COVERAGE PROBLEM AS AN INTEGER PROGRAM

Alan W. Williams, Robert L. Probert
School of Information Technology and Engineering,
University of Ottawa, Canada

Abstract A trend in software development is to assemble a system from a number of components. In many cases, the system is expected to function for multiple configurations of interchangeable components, leading to the problem of determining a set of system test configurations to fit a reasonable budget. One approach is to test all configurations that cover, for example, all two-way interactions. We investigate the feasibility of using an integer programming approach to solve the interaction test coverage problem exactly. We also examine the formulation to see if it provides insight into the NP-completeness of the interaction problem.

Key words: Interactions, components, coverage, system testing

1. INTRODUCTION

A common source of system faults is unexpected interactions between system components. The risk is magnified when there are a number of interchangeable components for each element in a system. A manufacturer of a system constructed from heterogeneous components would want to test as many of the potential system configurations as possible, to reduce the risk of interaction problems. However, the number of potential system configurations grows exponentially. The scenario in *Figure 1* has four parameters, each with three values. There are $3^4 = 81$ possible configurations.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35497-2_31
I. Schieferdecker et al. (eds.), *Testing of Communicating Systems XIV*

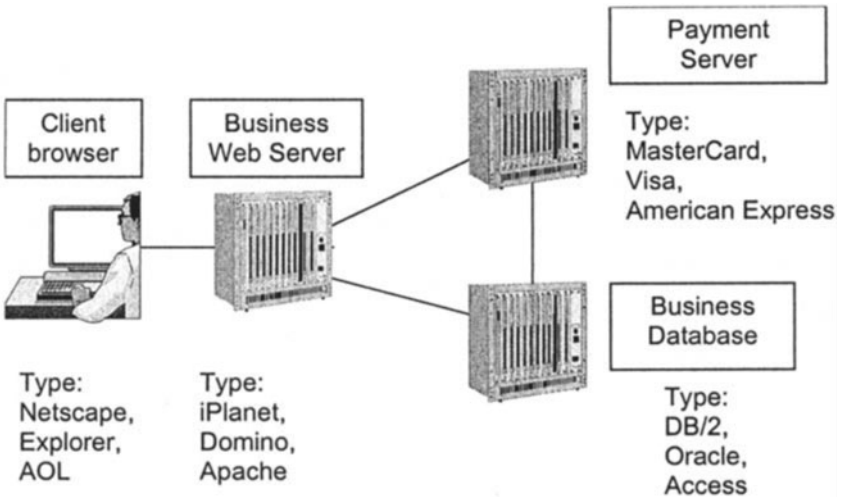

Figure 1. A System Test Scenario.

The system test suite for a single configuration may consist of a large number of tests. Even if there were only 30 tests for the system illustrated in Figure 1, running the entire test suite for every possible configuration would require running 2,430 tests. In practice, systems would have far more parameters and values, and larger test suites for a single configuration.

A system tester faces the constraints of time and money, and all possible configurations cannot be tested within any reasonable allotment. Furthermore, changing between configurations normally requires additional effort, either in physical re-configuration, or in modification of test cases. Therefore, reducing the number of system test configurations will produce significant savings. How, then, can the risk of interaction faults be managed in a realistic test plan?

One approach is to at least test for all two-way interactions among various system components. This leads to a reduced set of test configurations. The assumption is that the risk of an interaction among three or more components is balanced against the ability to complete system testing within a reasonable budget.

Previous work on determining a set of test configurations that covers two way interactions includes 0, 0, 0, 0, 0, and 0. Previous work where this method has been applied to various types of testing includes 0, 0, 0, 0, 0, 0, and 0. In this paper, we extend the work in 0 that formally defines interaction test coverage by investigating an approach that could solve the problem of how to find the smallest set of test configurations to achieve interaction coverage of a specified degree (for example, cover all 2-way

interactions). A fast algorithm that finds an approximate solution has been presented in 0; here, we try to solve the problem exactly.

This paper starts with a review of the interaction test coverage problem. We then show that the problem can be formulated as a {0,1} integer programming problem. We show that while the conversion can be done in that direction, a general {0,1} integer programming problem cannot be converted to an interaction test coverage problem. Thus, this relationship cannot be used to show that the interaction test coverage problem is NP-complete. (This is a problem that is still open. 0, 0, 0) Next, we discuss the feasibility of this approach in practice, and conclude with further avenues for investigation.

2. THE INTERACTION TEST COVERAGE PROBLEM

In this section, we set up the problem of selecting test configurations to cover parameter interactions. We subdivide the problem into two parts. The first part is the general problem of selecting from a set of discrete parameter values. In the second part, we look at the problem from the point of view of coverage of sets of interactions. The following is adapted from 0.

2.1 Basic definitions

We start by introducing terminology that will help to define precisely the test interaction coverage problem. The basic inputs to the problem are the number of parameters, and the number of values for each parameter. We begin by presenting notations for these quantities. Let:

p denote the number of independent system parameters.

n_i denote the number of possible values for the parameter with index i. We can assume, without loss of generality, that the parameters are ordered such that $n_1 \geq n_2 \geq \ldots \geq n_p$. In the special case where $n_1 = n_2 = \ldots = n_p$, the common number of values for each parameter will be denoted by n.

v_i denote a **selected indefinite** value in a specific test configuration for the parameter with index i.

1_3 denote a **selected definite** value in a specific test configuration (in this case, parameter 3 takes the value 1).

τ denote a test configuration, where $\tau = \{v_1, \ldots, v_p\}$.

T denote the set of all possible test configurations.

S denote the set of test configurations selected for execution.

d denote the **interaction degree**; the size of the subsets of parameter values for which we wish to detect unwanted interactions.

σ_d denote a subset of the parameter indices of size d; that is, we are choosing d out of p of the parameters.

Φ_d denote the set of all possible parameter subsets σ_d. The size of this set is $|\Phi_d| = \binom{p}{d}$.

The notation for definite and indefinite selected values allows us to keep track, using the subscript, of which parameter has been assigned a value in situations when we shall refer to a partial set of parameter values.

The property of parameter independence means that the selection of a particular value for one parameter affects neither the existence of other parameters, nor the selection of any other parameter values. This may not be the case in general. However, dependencies among parameters can be resolved by substituting a hybrid parameter that enumerates the legal combinations of the dependent parameters 0.

2.2 A general test coverage criterion

In practice, it is not usually possible to test all possible configurations, and therefore we have to select a subset for actual execution. The general test configuration selection problem is to find $S \subseteq T$ such that S satisfies a test coverage criterion.

Normally, an additional objective is to minimize $|S|$, the cardinality of the set of test configurations that satisfy the test coverage criterion. For the moment, we will leave the exact criterion that S satisfies undefined.

We want to find the set of test configurations with minimal size that will satisfy our (currently, undefined) test coverage criterion. Our goal is to define a test coverage criterion using a set of "test units" that need to be covered. We shall investigate what constitutes an interaction testing coverage unit in the next section.

2.3 Applying the criterion to interaction testing

To this point, all of the concepts defined and discussed in this section are generic in nature, in that they do not rely on any particular test selection strategy. In this section, we shall fill in the undefined test coverage criterion from the previous section, by introducing the concept of interaction elements.

The idea behind interaction elements is that they represent a "unit" of test coverage, in terms of system component interaction testing. Other types of test "units" used in other contexts (see, for example, 0) are as follows:

- Control flow testing: statements, branches, flow graph nodes, flow graph edges, conditions
- Data flow testing: definition-use pairs, IO-chains

In the context of system component interaction testing, the approach that we are taking is that there could be unwanted interactions among system components when a particular set of values is assigned to the set of parameters. This would be detected when a test suite is run using the configuration defined by the set of parameter values.

We shall take the view that an unwanted interaction is usually not caused by the particular values of the **entire** set of parameters, but by the values of only a (hopefully, small) **subset** of parameters. The philosophy of test coverage we shall adopt is to try and cover as many subsets of parameters as possible, where we shall limit ourselves to subsets of size $d \leq p$. These subsets will be referred to as interaction elements (or interaction elements of degree d, when referring to the cardinality of the subsets).

The aim is to reduce the number of test configurations to the point where the testing can be conducted with a feasible cost in time and money, and still have a good probability of detecting unwanted system interactions.

While it would be ideal to have $d = p$, it is shown in 0 that the number of test configurations to meet this test coverage criterion is be proportional to n^d. In practice, constraints of time and money will usually dictate that a small value of d, such as $d = 2$, be used. A study of the effectiveness of various values of d is presented in 0.

There is no particular order to the selection of parameters, so we shall use a set of d indices as opposed to an ordered d-tuple, when selecting a subset of parameters.

2.4 Interaction elements

Suppose that $\tau = \{v_1, \ldots, v_p\}$ is a given test configuration in T, and σ_d is a given index subset of $\{1, \ldots, p\}$. An **interaction element of degree *d***, denoted by $\chi(\sigma_d, \tau)$, is defined as $\chi(\sigma_d, \tau) = \{v_j \mid v_j \in \tau \text{ and } j \in \sigma_d\}$.

In other words, we are selecting a subset, of size d, of the assigned values for a given test configuration. An interaction element captures a single d-way interaction among the parameter values. The notation is chosen to emphasize that there are two parts to constructing an interaction element: selecting a subset of parameters, and that specific values are assigned to those parameters.

Because we have chosen a subset of the parameter indices, there are no repeated elements, and therefore an interaction element will not contain two values for the same parameter. This is a crucial property of a test configuration that we wish to preserve. The set notation is used to indicate that the ordering of the elements is not important. We can recover the parameter associations through the subscripts.

It is time for an example. Suppose that we want to select the values for parameters 1, 3, and 4 from the test configuration $\tau = \{5_1,4_2,3_3,2_4,1_5\}$. For $d = 3$, and $\sigma_3 = \{1,3,4\}$, we have $\chi(\sigma_3, \tau) = \{5_1,3_3,2_4\}$. The subscripts of the interaction element match the index set.

Having defined an interaction element, the next step is to look at the set of all possible interaction elements for specified values of p, n_i, and d. Let us denote this set by X_d, to highlight the dependence on d. The set X_d is:

$X_d = \{ \{ v_j \mid 1 \leq v_j \leq n_j, j \in \sigma_d, \} \mid \sigma_d \in \Phi_d \}$

That is, we take all possible index subsets of size d of the parameters. For each index subset, construct a set for every possible combination of d values for the associated parameters. The resulting collection is the set X_d.

Another way to look at the set X_d is consider it to be the set of all interaction elements contained within all possible test configurations. While the definition of interaction elements is dependent on a specific test configuration τ, the set of all possible interaction elements X_d is independent of the choice of test configurations.

For general values of d, the set X_d is the set of all possible interaction elements of degree d. The number of possible interaction elements is $|X_d| = \sum_{i_1=1}^{p-d} \cdots \sum_{i_d=i_{d-1}+1}^{p} (n_{i_1} \ldots n_{i_d})$. In the special case when $n = n_i$ for all i in the range $1 \leq i \leq p$ (that is, all parameters have n values), $|X_d| = \binom{p}{d} n^d$.

2.5 Test configurations as sets of interaction elements

We have previously defined concepts related to system test configuration selection and interaction elements. We can now bring these concepts together to define the interaction test coverage problem.

Suppose that we consider a test configuration in an alternative form: as a set of interaction elements of degree d, where d can be chosen arbitrarily to balance sufficient coverage with a feasible number of test configurations. Within the p parameter values of any test configuration, there are $\binom{p}{d}$ possible subsets of size d of parameter values.

Let $\tau = \{v_1, \ldots, v_p\}$ be a specific test configuration. As an alternative to treating a configuration τ to be a set of p values, a test configuration can be expressed (denoted τ') as a set of $\binom{p}{d}$ interaction elements of degree d:

$\tau' = \{ \chi(\sigma_d, \tau) \mid \sigma_d \in \Phi_d \}$

In other words, τ' is the set of interaction elements produced from all possible index subsets Φ_d for a single test configuration. Each interaction element must be an element of X_d, as X_d contains all possible interaction elements of degree d, over the entire range of values of the parameters.

Alternatively, τ' is the set of all subsets of size d of the assigned values for a specific test configuration. It captures the concept of a test configuration as a set of d-way interactions.

For example, suppose there are three parameters. The test configuration $\{1_1,1_2,1_3\}$ can also be considered as a set of three interaction elements of degree 2, namely $\{\{1_1,1_2\}, \{1_1,1_3\}, \{1_2,1_3\}\}$. In the latter form, we are considering this test configuration to represent the three pairs of 1's produced by selecting parameters, two at a time.

Clearly, each test configuration τ', constructed from an element of T, is a subset of X_d, the set of all interaction elements of degree d. Therefore, the entire set of potential test configurations, T, is a collection of subsets of X_d. This is a significant property that we shall incorporate into our test coverage criterion.

Let T be the set of all possible test configurations. Then, the set of all possible test configurations expressed as interaction elements of degree d, denoted T', is:

$T' = \{ \{ \chi(\sigma_d, \tau) \} \mid \sigma_d \in \Phi_d, \tau \in T \}$.

Each element of T' is a set of cardinality $\binom{p}{d}$, where each of those sets is in turn, a set of cardinality d. There is a one to one correspondence between elements of T', and elements of T, as each element in T' is constructed from an element of T.

Suppose that S is an arbitrary set of test configurations so that $S \subseteq T$. For each test configuration $\tau \in S$, let τ' be the equivalent test configuration expressed as a set of interaction elements of degree d. We can then denote as S' the set of all selected configurations τ' expressed as interaction elements. Since $S \subseteq T$, $S' \subseteq T'$.

2.6 The interaction test coverage problem

We can now define the *interaction test coverage problem*: the goal is to find $S \subseteq T$ (and thus $S' \subseteq T'$), such that every element in X_d belongs to at least one member of S'. In other words, we require that every interaction element be covered by a selected test configuration.

An additional objective is to minimize $|S|$, the cardinality of the set of selected test configurations. We want the minimum number of test configurations that will cover all d-wise combinations among parameter values.

This definition corresponds to other types of test coverage, where the goal is to cover all (or as large a percentage as possible) of some sort of test unit, within the set of selected test configurations. We are using interaction elements as test units for system interaction testing, as one would use control flow branches or definition-use associations in other type of test coverage criteria. Since we are assuming that the parameters are independent, all of our potential test configurations are feasible, and therefore, the test coverage criterion is to cover **all** interaction elements.

If a test coverage criterion based on interaction elements is used, then any interaction problem that is caused by an interaction of up to d specific values can be detected. Interactions of $d + 1$ or more values may not be detected. Thus, the criterion also defines the degree of risk inherent in a set of test configurations.

3. FORMULATION AS A {0,1} INTEGER PROGRAM

In this section, we shall re-formulate the interaction test coverage problem as a constraint-based problem. The result will become a {0, 1} integer programming problem.

Our objective in the interaction test coverage problem is to cover all of the interaction elements using a subset of all possible configurations. That is, given the set X_d of all interaction elements, and the set of all possible test configurations T, select $S \subseteq T$ as the set of selected test configurations. When each element of S is expressed as a set of interaction elements, the set S' is formed as the collection of all covered interaction elements. Then, every element of X_d must be in S'.

3.1 Set up the problem

Suppose that we construct an enumeration of the elements of T. The number of possible test configurations is $|T| = \prod_{i=1}^{p} n_i$. Let $z = |T|$. Furthermore, we will also construct an enumeration of the set of all possible interaction elements. The number of interaction elements is $|X_d| = \sum_{i_0=1}^{p-d+1} \cdots \sum_{i_{d-1}=i_{d-2}+1}^{p} \left(n_{i_0} \ldots n_{i_{d-1}}\right)$. Let $y = |X_d|$. In the special case where all parameters have the same number of values n, $z = n^p$ and $y = \binom{p}{d} n^d$.

Set up variables $\{x_1, \ldots, x_z\}$, where the value of $x_j = 1$ if configuration number j is selected (i.e. contained in S), and 0 if configuration number j is

not selected. Since the number of test configurations grows exponentially in the number of parameters, this will be a large set of variables for all but the smallest of test coverage problems.

Now, for each interaction element, determine the set of configurations that cover that interaction element. These will be all configurations where the d parameters in the index set for the interaction element are fixed and the remaining $p - d$ parameters can take any possible value. In the case where there are n values for each parameter, this means that there are n^{p-d} configurations that would cover this interaction element. Define an array of coefficients A such that $a_{ij} = 1$ if interaction element number i is covered by configuration number j, and 0 otherwise. For any i, there will be $p - d$ non-zero values of a_{ij}.

For each of the n^{p-d} configurations, we have a set of n^{p-d} variables representing whether or not the configuration is selected. Therefore, the sum of those variables represents the number of times the particular interaction element is covered. This sum is $\sum_{j=1}^{z} a_{ij} x_j$.

To meet our coverage goal, we require that the sum of those variables is greater than or equal to one, so that the interaction element is covered by at least one test configuration. This represents one constraint on our set of selected test configuration. Therefore, the set of y interaction elements will produce a set of y constraints of the form $\sum_{j=1}^{z} a_{ij} x_j \geq 1$, where $1 \leq i \leq y$.

Additionally, we would like to minimize the total number of configurations that are selected. That is, we would like $\sum_{j=1}^{z} x_j$ to be minimized.

Put together, this forms a {0, 1} integer-programming problem: minimize $\sum_{j=1}^{z} x_j$ subject to the constraints $\sum_{j=1}^{z} a_{ij} x_j \geq 1$, where $1 \leq i \leq y$.

At this point, a comment on the restriction of the values x_j to 0 or 1 is in order. Clearly, the values must be restricted to non-negative integers, as a test configuration cannot be "partially selected" (a fractional value) or "negatively selected". However, suppose that we relax the restriction on the values of x_j to be non-negative integers instead.

Furthermore, suppose that in a minimized sum $\sum_{j=1}^{z} x_j$, there is a value $x_q > 1$. Since all of the constraints are of the form $\sum_{j=1}^{z} a_{ij} x_j \geq 1$, where

$a_{ij} \in \{0, 1\}$, we could replace the value of x_q by 1, and the constraint would still be satisfied. Furthermore, since all of the x_j are non-negative integers, the sum $\sum_{j=0}^{z-1} x_j$ would be reduced by the amount $x_q - 1$, which contradicts the assumption that the sum was minimized. Therefore, in the minimized sum, all of the $x_j \in \{0, 1\}$.

The result is that the restriction $x_j \in \{0, 1\}$ does not exclude any potential minimal solutions. The restriction may be useful in practice, as we can provide an integer-programming solver with the information that the variables are binary in nature. This allows the integer-programming solver to discard potential solutions more quickly. (Note that either of these are still much more difficult to solve as compared with a linear programming problem.)

Here is an example. Suppose that there are three parameters, and there are two possible values for each parameter. There are $2^3 = 8$ possible test configurations, and $\binom{3}{2}2^2 = 12$ interaction elements to be covered. We define a set of variables $\{x_1, \ldots, x_8\}$ such that $x_j \in \{0, 1\}$. We will have $x_j = 1$ if configuration j is selected, and 0 otherwise. *Figure 2* illustrates the resulting {0,1} integer programming problem.

In *Figure 2*, the sets of test configurations are listed in the column headings, and the sets of interaction elements are listed in the row headings.

Interaction Elements	Configurations {1,1,1}	{1,1,2}	{1,2,1}	{1,2,2}	{2,1,1}	{2,1,2}	{2,2,1}	{2,2,2}	
$\{1_1,1_2\}$	x_1	$+x_2$							≥ 1
$\{1_1,1_3\}$	x_1		$+x_3$						≥ 1
$\{1_2,1_3\}$	x_1				$+x_5$				≥ 1
$\{1_1,2_2\}$			x_3	$+x_4$					≥ 1
$\{1_1,2_3\}$		x_2		$+x_4$					≥ 1
$\{1_2,2_3\}$		x_2				$+x_6$			≥ 1
$\{2_1,1_2\}$					x_5	$+x_6$			≥ 1
$\{2_1,1_3\}$					x_5		$+x_7$		≥ 1
$\{2_2,1_3\}$			x_3				$+x_7$		≥ 1
$\{2_1,2_2\}$							x_7	$+x_8$	≥ 1
$\{2_1,2_3\}$						x_6		$+x_8$	≥ 1
$\{2_2,2_3\}$				x_4				$+x_8$	≥ 1

Figure 2. Formulation of a test coverage problem as an integer program.

If we minimize $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8$, subject to $x_1 + x_2 \geq 1$, $x_1 + x_3 \geq 1$, $x_1 + x_5 \geq 1$, $x_3 + x_4 \geq 1$, $x_2 + x_4 \geq 1$, $x_2 + x_6 \geq 1$, $x_5 + x_6 \geq 1$, $x_5 + x_7 \geq 1$, $x_3 + x_7 \geq 1$, $x_7 + x_8 \geq 1$, $x_6 + x_8 \geq 1$, $x_4 + x_8 \geq 1$, we will have obtained the set of test configurations of minimal cardinality that covers all interaction elements.

3.2 Solving the integer program

For experimental purposes, a linear programming solver called "lp_solve" 0 written by Michel Berkelaar was used. Lp_solve is available as free-ware from Eindhoven University. The input file format is straightforward, given the above discussion:

```
min: x0 + x1 + x2 + x3 + x4 + x5 + x6 + x7;
x0 + x1 >= 1;
x1 + x2 >= 1;
x1 + x3 >= 1;
x1 + x5 >= 1;
x3 + x4 >= 1;
```

```
x2 + x4 >= 1;
x5 + x6 >= 1;
x5 + x7 >= 1;
x3 + x7 >= 1;
x7 + x8 >= 1;
x6 + x8 >= 1;
x4 + x8 >= 1;
int x1, x2, x3, x4, x5, x6, x7, x8;
```

The first line represents the objective function, and indicates that it should be minimized. The constraints are listed, and the final line indicates that variables `x1` through `x8` should be restricted to integers. An implied constraint is that all variables are greater than or equal to zero.

The output produced by lp_solve for the above input file is:

```
CPU Time for parsing input: 0.01s (0.01s total since
program start)
CPU Time for solving: 0s (0.01s total since program
start)
Value of objective function: 4
x1                    1
x2                    0
x3                    0
x4                    1
x5                    0
x6                    1
x7                    1
x8                    0
```

The value of the objective function is 4, which is the sum of `x1` through `x8`. This indicates that four test configurations are sufficient to provide coverage of all interaction elements. The specific values of the solution variables indicate that the test configurations {1, 1, 1}, {1, 2, 2}, {2, 1, 2}, and {2, 2, 1} should be selected. The configurations can be determined quickly by representing the variable number (plus 1, to allow for indexing from 1 instead of 0) in binary, and increasing each value by 1: for example, `x4` is binary 101, which indicates {2, 1, 2}.

By formulating the interaction test coverage problem as a {0, 1} integer programming problem, solving the resulting integer program exactly will produce the best possible solution.

3.3 Difficulties with the approach

We run into problems because the number of variables required grows exponentially. If there is the same number of values, n, for each parameter, we have the following results:

The row dimension will be $\binom{p}{d} n^d$, which is the number of interaction elements. This is somewhat manageable, since the power in the exponent is only d.

The column dimension will be n^p, which is the number of configurations. This will be extremely large in most cases.

The number of non-zero a_{ij} per row will be n^{p-d}

The number of non-zero a_{ij} will be $\binom{p}{d} n^p$

To get an idea of how quickly this becomes infeasible to calculate in practice, here are some experimental results using the lp_solve tool:

Table 1. Results of running lp_solve on test configuration problems.

# parms	# values per parm	# constraints	# variables	Result: # configs	Run time (s)
3	2	12	8	4	<0.01
4	2	24	16	5	0.01
5	2	40	32	6	0.70
6	2	60	64	6	16.57
7	2	84	128	6	441.21
4	3	54	81	9	0.08
5	3	90	243	13*	*

The * in *Table 1* indicates the best possible solution found at the point when the process was killed after running for about 6.5 hours. Some results provided by other users of lp_solve, (included as documentation in the distribution) indicate lp_solve will have difficulties for problems that have more than about 100 **integer** variables. The above results support this statement.

3.4 Is the interaction test coverage problem NP-complete?

We have reformulated the interaction test coverage problem as a {0, 1} integer programming problem, which is NP-complete 0. We now present an argument why a general {0, 1} integer programming problem does not reduce directly by this mapping to a test configuration problem.

In a general {0, 1} integer programming problem, we could have, for example, the following set of constraints:

Constraint 1: $x_1 + x_2 + x_3 \geq 1$

Constraint 2: $x_4 + x_5 \geq 1$

Constraint 3: $x_6 + x_7 \geq 1$

Constraint 4: $x_1 + x_4 + x_7 \geq 1$

Constraint 5: $x_2 + x_5 + x_1 \geq 1$

We have seven variables and five constraints and the key point that seven is a prime number. Now, the number of variables represents the number of configurations, which is $\prod_{i=1}^{p} n_i$. However, if we have a prime number as the number of configurations, then there can be only one parameter with more than one value. This parameter must have seven possible values.

Now, the number of interaction elements is $|X_d| = \sum_{i_1=1}^{p-d} \ldots \sum_{i_d=i_{d-1}+1}^{p} (n_{i_1} \ldots n_{i_d})$, and this is the number of constraints in an interaction test coverage problem. However, we know of the existence of at least one parameter with seven values, and therefore this sum is at least seven. We cannot have any terms of this sum less than zero, since this would represent a parameter with a negative number of values, and that is definitely not an interaction test coverage problem. However, our integer program has only five constraints, and therefore cannot represent the number of interaction elements for any problem with seven variables.

Thus, we have found a counter-example to the hypothesis that a general {0,1} integer programming problem can be represented directly as an interaction test coverage problem.

Therefore, we cannot reduce a general {0, 1} integer programming problem to a test configuration problem, and therefore we cannot obtain a result as to whether the interaction test coverage problem is NP-complete using this approach.

There is regularity to how the 0 and 1 coefficients appear in the constraint grid, of which the integer programming formulation does not take advantage. That regularity is also enough to guarantee that there is a feasible minimum solution to the integer-programming problem constructed out of a

test configuration problem. There has to be at least one feasible solution since we are guaranteed to cover every interaction element by selecting every possible test configuration. Of course, the minimum ought to be much less than that, but we have shown the existence of at least one solution, and therefore a minimum has to exist.

4. CONCLUSIONS AND FURTHER WORK

We have shown that the interaction test coverage problem can be solved exactly by formulating the problem as a {0,1} integer program. Unfortunately, this leads to difficulties in practice because the latter is NP-complete. The experiments we have run show that only the very smallest of interaction test coverage problems can be solved with this approach.

We have also shown the reverse mapping is not possible; a general {0,1} integer programming problem cannot always be converted to an interaction test coverage problem. Therefore, we cannot establish that the interaction test coverage problem is NP-complete by this route.

For further work, one avenue of investigation is whether a linear programming approximation will produce useful results. Our initial experiments indicate that such an approximation is possible, but the results do not appear to be as good as the recursive covering array construction of 0.

Other work is needed in extending the results of 0 to cover interactions of degree higher than 2, and to investigate recursive covering array construction for parameters with varying numbers of values.

REFERENCES

Beizer, B. *Software Testing Techniques*, Second Edition, Van Nostrand Reinhold, New York NY USA, 1990.

Berkelaar, M., "lp_solve," version 3.0. Linear programming solver developed at Eindhoven University, available at ftp://ftp.ics.ele.tue.nl/pub/lp_solve

Brownlie, R., Prowse, J., and Phadke, M.S. Robust Testing of AT&T PMX/StarMail using OATS. *AT&T Technical Journal 71*, 3 (May/June 1992), 41-47.

Burroughs, K., Jain, A., and Erickson, R.L. Improved Quality of Protocol Testing Through Techniques of Experimental Design. In *Proceedings of Supercomm/ICC '94*, (1994), 745-752.

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

Cohen, D.M., Dalal, S.R., Parelius, J., and Patton, G.C. The Combinatorial Approach to Automatic Test Generation. *IEEE Software 13*, 5 (Sept. 1996), 83-88.

Crescenzi, P., Kann, V., editors. A compendium of NP-complete problems. On-line at http://www.f.kth.se/~viggo/problemlist/compendium.html

Dunietz, I.S., Ehrlich, W.K., Szablak, B.D., Mallows, C.L., and Iannino, A. Applying Design of Experiments to Software Testing. In *Proceedings of the 19th International Conference on Software Engineering (ICSE '97)*, (Boston MA USA, 1997), 205-215.

Karpinski, M. Polynomial Time Approximation Schemes for Some Dense Instances of NP-Hard Optimization Problems. In Proceedings of the 1st Symposium on Randomization and Approximation Techniques in Computer Science, Lecture Notes in Computer Science 1269, Springer-Verlag, 1997, 1-14.

Karpinski, M. and Zelikovsky, A. Approximating Dense Cases of Covering Problems. ECCC Technical Report TR97-004. On-line at ftp://eccc.uni-trier.de/pub/eccc/reports/1997/TR97-004/index.html.

Perkinson, W.B. A Methodology for Designing and Executing ISDN Feature Tests Using Automated Test Systems. In *Proc. of IEEE GLOBECOMM '92,* (1992).

Ryu, J., Kim, M., Kang S., and Seol, S. Interoperability Test Suite Generation for the TCP Data Part Using Experimental Design Techniques. In *Proceedings of the 13th International Conference on the Testing of Communicating Systems (Testcom 2000),* (Ottawa ON Canada, 2000), 127-142.

Stevens, B. and Mendelsohn, E. Efficient Software Testing Protocols. In *Proceedings of the 8th IBM Centre for Advanced Studies Conference (CASCON '98),* (Toronto ON, 1998), 279-293.

Williams, A.W., and Probert, R.L. A Practical Strategy for Testing pair-wise Coverage of Network Interfaces. In *Proceedings of the 7th International Conference on Software Reliability Engineering (ISSRE '96)*, (White Plains NY USA, 1996), 246-254.

Williams, A.W. Determination of Test Configurations for Pair-Wise Interaction Coverage. In *Proceedings of the 13th International Conference on the Testing of Communicating Systems (Testcom 2000),* (Ottawa ON Canada, 2000), 59-74.

Williams, A.W., and Probert, R.L. A Measure of Component Interaction Test Coverage. In *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications (AICCSA 2001),* (Beirut Lebanon, 2001), 304-311.