

# Generating Small Combinatorial Test Suites to Cover Input-Output Relationships

Christine T. Cheng   Adrian Dumitrescu   Patrick J. Schroeder  
Department of Computer Science  
University of Wisconsin–Milwaukee, Milwaukee, WI 53211, USA.  
{ccheng, ad, pats}@cs.uwm.edu

February 13, 2004

## Abstract

In this paper, we consider a problem that arises in black box testing: generating small test suites (i.e., sets of test cases) where the combinations that have to be covered are specified by input-output parameter relationships of a software system. That is, we only consider combinations of input parameters that affect an output parameter, and we do not assume that the input parameters have the same number of values. To solve this problem, we revisit the *greedy algorithm* for test generation and show that the size of the test suite it generates is within a logarithmic factor of the optimal. Unfortunately, the algorithm’s main weaknesses are its time and space requirements for construction. To address this issue, we present a *problem reduction technique* that makes the greedy algorithm and possibly any other test suite generation method more efficient if the reduction in size is significant. We discuss its merits and limitations, and evaluate the technique on actual software systems.

## 1 Introduction

Software testing remains an important topic in software engineering. A recent report generated for the National Institute of Standards and Technology (NIST) found that software defects cost the U.S. economy 59.9 billion dollars annually [19]. While current technologies cannot hope to remove all errors from software, the report goes on to estimate that 22 billion dollars could be saved through earlier and more effective defect detection.

Black-box testing is a type of software testing that ensures a program meets its specification from a behavioral or functional perspective. The number of possible black-box test cases for any non-trivial software application is extremely large. The challenge in testing is to reduce the number of test cases to a subset that can be executed with available resources and can also exercise the software adequately so that majority of software defects are exposed. One popular method for performing black-box testing is the use of combinatorial covering designs [9, 4, 6] based on techniques developed in designing experiments. These designs correspond to test suites (i.e., a set of test cases) that *cover*, or execute, combinations of input parameters in a systematic and effective way, and are most applicable in testing data-driven systems where the manipulation of data inputs and the relationship between input parameters is the focus of testing. As pointed out by Dunietz, *et al.* [9], a technical challenge that remains in applying this promising technique in software testing

is the construction of covering designs. There are two issues that need to be considered: the first and perhaps more important one is the size of the covering design since it dictates the number of test cases, and consequently, the amount of resources needed to test a software system; the second is the time and space requirements of the construction itself.

A great deal of research work has been devoted to generating small test suites [2, 3, 12, 13, 7, 14, 16, 22]. Most researchers focus on *uniform coverage* of input parameters with *uniform ranges*; i.e., they consider test suites that cover all  $t$ -wise combinations of the input parameters<sup>1</sup> for some integer  $t$  and the input parameters are assumed to have the same number of values. While such test suites apply to a large number of situations, in practice not all  $t$ -wise combinations of input parameters have equal priority in testing [2, 17], nor are all parameter domains of the same size [3, 14]. Testers often prioritize combinations of input parameters that influence a system’s output parameters over those that do not [2, 3, 8, 15, 20]. Determining which set of input parameters influence a system’s output parameters can be accomplished using existing analyses [8, 15, 20] or can be discovered in the process of determining the expected result of a test case.<sup>2</sup>

Given a system’s *input-output relationships*, a tester may wish to create a test suite that covers only those combinations of input parameters that influence the program outputs in lieu of a test suite that covers all  $t$ -wise combinations of uniform input parameters. How should such a test suite be generated so that its size is small and its construction is efficient? To this end, we revisit the *greedy algorithm* for test generation which was first analyzed by Cohen *et al.* [4] in the context of  $t$ -wise coverage of input parameters with the same ranges. Not only is the greedy algorithm applicable in this general setting, but we will also prove that the size of the test suite it generates is relatively small – it is within a logarithmic factor of the optimal. Unfortunately, its main weaknesses are that its time and space requirements become prohibitive as the number of input parameters increases. To address this issue, we present a *problem reduction technique* that makes the greedy algorithm and possibly any other test suite generation method more efficient if the reduction in size is significant.

In section 2, we define our terms and discuss the greedy algorithm further. In section 3, we present our problem reduction technique and analyze its merits. Initially, we assume that the input parameters have the same number of values; later on, we suggest several approaches on how to extend our technique when the assumption is no longer true. We then evaluate the technique on actual software systems, and conclude in section 4.

## 2 Constructing small combinatorial test suites

Assume that a software system has  $k$  input parameters which influence  $n$  output parameters. Let  $\mathcal{I} = \{I_1, \dots, I_k\}$  be the set of the input parameters where each parameter  $I_s$  has  $l_s$  values. For simplicity, we represent the  $l_s$  values as the numbers  $1, 2, \dots, l_s$ . Let  $O_j$  be the subset of input parameters that affect the  $j$ th output parameter, and  $\mathcal{O} = \{O_1, \dots, O_n\}$ . If all the parameters have the same number of values, we say that the parameters have *uniform ranges*; if  $\mathcal{O}$  consists of all size- $t$  subsets of  $\mathcal{I}$ , we say that the parameters have *uniform  $t$ -wise coverage*. As we mentioned earlier, we will assume that the parameters need not have uniform ranges nor uniform coverage.

<sup>1</sup>Suppose  $I_1, I_2, \dots, I_k$  are the input parameters and  $I_s$  has  $l_s$  values for  $s = 1, \dots, k$ . A  $t$ -wise combination of the input parameters is a  $t$ -tuple  $(v_{s_1}, \dots, v_{s_t})$  where  $v_{s_j}$  is a value for parameter  $I_{s_j}$  for  $j = 1, \dots, t$ .

<sup>2</sup>In order to determine the expected result of a test case, the tester must know which input parameters influence, or affect the computation of the system’s output parameters.

A test case  $T$  for the problem instance  $(\mathcal{I}, \mathcal{O})$  is a  $k$ -tuple  $(t_1, t_2, \dots, t_k)$  where  $t_s$  is a value of  $I_s$  for each  $s$ . Let  $O_j = \{I_{s_1}, I_{s_2}, \dots, I_{s_r}\}$ . The test case  $T$  covers one combination of  $O_j$ :  $t_{s_1}, t_{s_2}, \dots, t_{s_r}$ . A test suite  $\mathcal{T}$  for  $(\mathcal{I}, \mathcal{O})$  is a set of test cases for the problem instance that covers all the  $l_{s_1} \times l_{s_2} \times \dots \times l_{s_r}$  combinations of  $O_j$  for each  $O_j \in \mathcal{O}$ . In our discussion, we shall arrange the test cases of  $\mathcal{T}$  as columns of a  $k \times |\mathcal{T}|$  array whose rows are indexed by  $I_1, \dots, I_k$ . An important goal in software testing is to construct  $\mathcal{T}$  so that its size is as small as possible.

Finding optimal test suites for arbitrary problem instances, however, is difficult. For instance, Seroussi and Bshouty [21] has shown that deciding if an arbitrary  $(\mathcal{I}, \mathcal{O})$  can be covered by a test suite of size four is NP-complete – even in the simple instance when all the parameters in  $\mathcal{I}$  are binary, and all the sets in  $\mathcal{O}$  have size two. Next, we analyze the following greedy algorithm for test suite generation: at each iteration, pick the unused test case that covers the maximum number of uncovered combinations. Cohen, *et al.* [4] were the first to obtain an upper bound on the number of test cases the greedy algorithm generates in the context of pairwise coverage of all input parameters with uniform ranges. They also generalized the bound when the input parameters have uniform  $t$ -wise coverage. Their result is the following:

**Theorem 2.1** [4] *Let  $\mathcal{I}$  consist of  $k$  parameters, each of which has  $l$  values. If all the pairwise combinations of the parameters in  $\mathcal{I}$  have to be covered then the greedy algorithm will generate a test suite whose size is at most  $\lceil l^2(\log l^2 + \log \binom{k}{2}) \rceil = O(l^2(\log l + \log k))$ .*

To extend this result for the general setting, we consider the *set-covering problem*. An instance of the set-covering problem consists of a finite set  $X$  and a collection  $\mathcal{F}$  of subsets of  $X$ . The goal is to find the smallest sub-collection  $C \subseteq \mathcal{F}$  so that every element in  $X$  is covered by or included in some set in  $C$ . The set-covering problem is NP-hard [11]. To find a sub-collection  $C$ , we can also use the same greedy algorithm we just described above. This time, at each iteration, the algorithm picks the unused set that covers the most remaining uncovered elements in  $X$ . It is known that the size of the resulting set cover is at most  $(1 + \ln |S_{\max}|)$  times the optimal, where  $S_{\max}$  is the largest set in  $\mathcal{F}$ , and  $\ln$  is the natural log [5].

Let us now cast the test generation problem as a set-covering problem. Let  $X$  consist of all combinations of  $(\mathcal{I}, \mathcal{O})$  that have to be covered. For example if  $O_j = \{I_{s_1}, \dots, I_{s_r}\}$ , all  $l_{s_1} \times \dots \times l_{s_r}$  combinations of  $O_j$  are in  $X$ . Let us also represent each test case  $T$  of  $(\mathcal{I}, \mathcal{O})$  as a set which consists of all the combinations it covers in  $X$ , and let  $\mathcal{F}$  consist of all the possible test cases. Since each test case covers exactly one combination of  $O_j$  for  $j = 1, \dots, n$ , every set in  $\mathcal{F}$  has size  $n$ . Once the transformation we have described is complete, finding an optimal test suite for  $(\mathcal{I}, \mathcal{O})$  is the same as finding an optimal set cover for  $(X, \mathcal{F})$ . Consequently, we have the following result:

**Theorem 2.2** *Suppose a software system has  $k$  input parameters and  $n$  output parameters and the combinations that need to be covered are described by the instance  $(\mathcal{I}, \mathcal{O})$ . The greedy algorithm for test generation produces a test suite whose size is at most  $(1 + \ln n) \times OPT$ , where  $OPT$  is the size of the optimal test suite.*

We emphasize that the approximation factor in our theorem depends only on the number of output parameters, and not on the number of input parameters nor their ranges. In the case when  $\mathcal{O}$  consists of all 2-subsets of  $\mathcal{I}$  (as in the situation described in Theorem 2.1),  $n = \binom{k}{2}$  so our theorem states that the number of test cases produced by greedy is at most  $(1 + \ln \binom{k}{2}) \times OPT$ .

### 3 A problem reduction technique

The greedy algorithm has many attractive features: it is simple to implement, flexible, and generates a relatively small number of test cases. However, its time and space requirements grow exponentially as the number of input parameters increases. This is due to the fact that at each iteration it needs to find the unused test case that covers the most number of uncovered combinations. If there are ten parameters with five values each, for example, then at each iteration it must check the coverage of approximately nine million test cases. It is this issue that we seek to address in this section. One obvious approach is to design a faster (and a more space-efficient) algorithm for generating test suites. This was taken by Cohen, *et al.* [4] where, instead of doing a brute force search at each iteration, they implemented a greedy random heuristic that finds an unused test case that covers a large (but perhaps not the maximum) number of uncovered combinations. Our approach, on the other hand, is to make test generation more efficient via *problem reduction*. That is, given  $(\mathcal{I}, \mathcal{O})$ , we reduce this instance to  $(\mathcal{I}', \mathcal{O}')$  so that (i)  $|\mathcal{I}'| \leq |\mathcal{I}|$  and  $|O'_j| \leq |O_j|$  for each  $j$ , and (ii) *any* test suite  $\mathcal{T}'$  for  $(\mathcal{I}', \mathcal{O}')$  can be transformed efficiently to a test suite  $\mathcal{T}$  for  $(\mathcal{I}, \mathcal{O})$ . If such a reduction is possible, then we just have to find a test suite for  $(\mathcal{I}', \mathcal{O}')$  using the greedy algorithm or any other test generation method. Since the input to these methods are now smaller, we expect them to run faster and require less space. Indeed, the smaller  $(\mathcal{I}', \mathcal{O}')$  is with respect to  $(\mathcal{I}, \mathcal{O})$ , the greater will be the gain in efficiency. First, we mention two simple reductions.

R1: If  $O_i \subseteq O_j$ , delete  $O_i$  from  $\mathcal{O}$ . This reduction is valid because any test suite that covers all the combinations of  $O_j$  also covers all the combinations of its subsets.

R2: If  $I_r$  has only one value, delete  $I_r$  from  $\mathcal{I}$  and from all  $O_j$ 's that contain  $I_r$ . Suppose  $\mathcal{T}'$  is a test suite for the reduced problem instance  $(\mathcal{I}', \mathcal{O}')$ . For each test case  $T' \in \mathcal{T}'$ , create a test case  $T$  for the original problem instance by setting  $I_r$  equal to its only value and all other parameters to their values in  $T'$ . Since the test cases of  $\mathcal{T}'$  cover all combinations of  $O'_j$ , the corresponding test cases for the original problem must cover all combinations of  $O'_j \cup \{I_r\}$ .

For now, we will assume that the parameters in  $\mathcal{I}$  have the same number of values  $l > 1$ . Later on, we will extend our technique to parameters that have non-uniform ranges. Our general technique for problem reduction finds a *base set of parameters*  $\mathcal{I}'$ , which is a subset of  $\mathcal{I}$ . Each  $I_r \in \mathcal{I}$  is then mapped to a parameter in  $\mathcal{I}'$ , say  $I_s$ , so that in creating a test case for the original problem instance, the value of  $I_r$  is set equal to the value of  $I_s$ . One implication of this mapping is that if  $I_r$  and  $I_s$  influence the same output, say  $O_j$ , then  $I_r$  cannot be mapped to  $I_s$ ; otherwise, some combinations of  $O_j$  will not be covered. We use a graph to model this restriction.

Let  $G$  be the graph whose vertices correspond to the  $k$  input parameters of the system. Two nodes are adjacent if and only if their corresponding input parameters together influence some output parameter; i.e., the input parameters are elements of some  $O_j$  in  $\mathcal{O}$ . Let us call  $G$  the *relationship graph* of the problem instance  $(\mathcal{I}, \mathcal{O})$ . We now construct the mapping between parameters, and generate the test cases in the following manner:

*Step 1:* Properly color  $G$  (using as few colors as you can); i.e., assign a color to each node of  $G$  so that no two adjacent nodes are assigned the same color. Let  $c$  be the number of colors used.

*Step 2:* For each color  $s$ , pick a parameter  $I'_s$  whose corresponding node in  $G$  is colored  $s$ . Let  $\mathcal{I}' = \{I'_1, I'_2, \dots, I'_c\}$ .

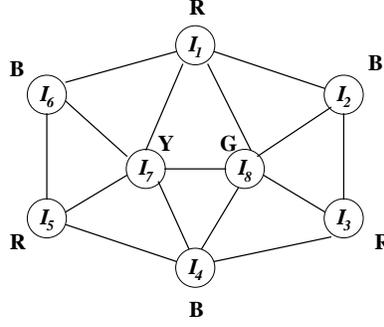


Figure 1: The relationship graph for the problem instance where  $\mathcal{I} = \{I_1, I_2, \dots, I_8\}$  and  $\mathcal{O} = \{\{I_1, I_2, I_8\}, \{I_2, I_3\}, \{I_3, I_4, I_8\}, \{I_4, I_5, I_7\}, \{I_5, I_6\}, \{I_1, I_6, I_7\}, \{I_7, I_8\}\}$ .

*Step 3:* For each  $O_j$ , construct  $O'_j$  where  $I'_s$  is an element of  $O'_j$  if and only if one of the parameters in  $O_j$  was colored with  $s$ . (That is, map every parameter colored  $s$  to  $I'_s$ .) Since no two parameters in  $O_j$  are assigned the same color because they are adjacent in  $G$ , it must be the case that  $|O_j| = |O'_j|$ , for all  $j$ . Apply reduction R1 to  $\{O'_1, \dots, O'_n\}$  and call the resulting set  $\mathcal{O}'$ .

*Step 4:* Construct a test suite  $\mathcal{T}'$  for  $(\mathcal{I}', \mathcal{O}')$ .

*Step 5:* Finally, for each test case  $T' \in \mathcal{T}'$ , construct a corresponding test case  $T$  for  $(\mathcal{I}, \mathcal{O})$  in the following way: for each  $I_r$ , if  $I_r$  was colored  $s$  then set its value in  $T$  equal to the value of  $I'_s$  in  $T'$ . Let  $\mathcal{T}$  denote the resulting test suite for  $(\mathcal{I}, \mathcal{O})$ . Note that  $|\mathcal{T}| = |\mathcal{T}'|$ .

*An example.* Let us now illustrate the reduction technique for the following example. Let  $\mathcal{I} = \{I_1, I_2, \dots, I_8\}$  and  $l = 2$ . Let  $\mathcal{O} = \{\{I_1, I_2, I_8\}, \{I_2, I_3\}, \{I_3, I_4, I_8\}, \{I_4, I_5, I_7\}, \{I_5, I_6\}, \{I_1, I_6, I_7\}, \{I_7, I_8\}\}$ . The relationship graph  $G$  is shown in Figure 1. We color  $G$  with four colors: red (R), blue (B), yellow (Y), green (G). Let us pick one parameter for each color:  $I_1$  for red,  $I_2$  for blue,  $I_7$  for yellow and  $I_8$  for green. Hence,  $\mathcal{I}' = \{I_1, I_2, I_7, I_8\}$ . Let us now replace the parameters in each  $O_j$  with their corresponding parameters in  $\mathcal{I}'$ :  $\{I_1, I_2, I_8\} \rightarrow \{I_1, I_2, I_8\}$ ,  $\{I_2, I_3\} \rightarrow \{I_2, I_1\}$ ,  $\{I_3, I_4, I_8\} \rightarrow \{I_1, I_2, I_8\}$ ,  $\{I_4, I_5, I_7\} \rightarrow \{I_2, I_1, I_7\}$ ,  $\{I_5, I_6\} \rightarrow \{I_1, I_2\}$ ,  $\{I_1, I_6, I_7\} \rightarrow \{I_1, I_2, I_7\}$ , and  $\{I_7, I_8\} \rightarrow \{I_7, I_8\}$ . Applying reduction R1,  $\mathcal{O}' = \{\{I_1, I_2, I_8\}, \{I_1, I_2, I_7\}, \{I_7, I_8\}\}$ . The table below contains a test suite  $\mathcal{T}'$  for  $(\mathcal{I}', \mathcal{O}')$ .

$I_1$	2	2	2	1	1	1	2	1
$I_2$	2	2	1	2	1	2	1	1
$I_7$	2	1	2	2	2	1	1	1
$I_8$	2	1	1	1	1	2	2	2

Next, we transform  $\mathcal{T}'$  into a test suite  $\mathcal{T}$  for  $(\mathcal{I}, \mathcal{O})$  according to step 5. Thus, the rows corresponding to  $I_3$  and  $I_5$  are identical to that of  $I_1$  because all three parameters were colored red, while the rows corresponding to  $I_4$  and  $I_6$  are identical to that of  $I_2$  because all three parameters were colored blue.

$I_1$	2	2	2	1	1	1	2	1
$I_2$	2	2	1	2	1	2	1	1
$I_3$	2	2	2	1	1	1	2	1
$I_4$	2	2	1	2	1	2	1	1
$I_5$	2	2	2	1	1	1	2	1
$I_6$	2	2	1	2	1	2	1	1
$I_7$	2	1	2	2	2	1	1	1
$I_8$	2	1	1	1	1	2	2	2

**Theorem 3.1** *At the end of step 5,  $\mathcal{T}$  is a test suite for  $(\mathcal{I}, \mathcal{O})$ . That is,  $\mathcal{T}$  covers all combinations of all sets in  $\mathcal{O}$ .*

**Proof.** Let  $O_j \in \mathcal{O}$  and  $(v_1, v_2, \dots, v_r)$  be a particular combination of  $O_j$ . At the end of step 3, either  $O'_j \in \mathcal{O}'$ , or  $O'_j$  is a proper subset of some set in  $\mathcal{O}'$ . In either case, since  $\mathcal{T}'$  is a test suite for  $(\mathcal{I}', \mathcal{O}')$  at step 4,  $\mathcal{T}'$  covers all combinations of  $O'_j$ . Let  $T'$  be the test case in  $\mathcal{T}'$  that covers the combination  $(v_1, v_2, \dots, v_r)$  of  $O'_j$ . The corresponding test case  $T$  of  $T'$  at step 5 must then cover the same combination for  $O_j$ . Since we chose  $O_j$  and its combination arbitrarily, we have proven that  $\mathcal{T}$  does cover all combinations of all sets in  $\mathcal{O}$ .  $\square$

**Theorem 3.2** *Suppose  $(\mathcal{I}, \mathcal{O})$  was reduced to  $(\mathcal{I}', \mathcal{O}')$  using steps 1 to 3 of our algorithm. If  $\mathcal{T}_{opt}$  and  $\mathcal{T}'_{opt}$  are optimal test suites for  $(\mathcal{I}, \mathcal{O})$  and  $(\mathcal{I}', \mathcal{O}')$  respectively, then  $|\mathcal{T}_{opt}| \leq |\mathcal{T}'_{opt}|$ . In addition, if  $\mathcal{O}' \subseteq \mathcal{O}$ , then  $|\mathcal{T}_{opt}| = |\mathcal{T}'_{opt}|$ .*

**Proof.** According to Theorem 3.1, if  $\mathcal{T}'_{opt}$  is the test suite chosen for  $(\mathcal{I}', \mathcal{O}')$ , then step 5 of our algorithm transforms it into a valid test suite for  $(\mathcal{I}, \mathcal{O})$  whose size is  $|\mathcal{T}'_{opt}|$ . Hence,  $|\mathcal{T}'_{opt}| \geq |\mathcal{T}_{opt}|$ . On the other hand, when  $\mathcal{O}' \subseteq \mathcal{O}$ ,  $\mathcal{T}_{opt}$  can also be transformed into a test suite for  $(\mathcal{I}', \mathcal{O}')$ . Arrange the test cases of  $\mathcal{T}_{opt}$  as columns of a  $k \times |\mathcal{T}_{opt}|$  array whose rows are indexed by the parameters of  $\mathcal{I}$ . Delete all rows corresponding to the input parameters *not* in  $\mathcal{I}'$ . Since  $\mathcal{T}_{opt}$  covers all combinations of all the sets in  $\mathcal{O}'$ , the new test suite must do so likewise. Hence,  $|\mathcal{T}_{opt}| \geq |\mathcal{T}'_{opt}|$  and the theorem follows.  $\square$

The next corollary follows immediately from Theorem 3.2.

**Corollary 3.3** *Suppose  $\mathcal{O}' \subseteq \mathcal{O}$ . Let  $\mathcal{T}'$  and  $\mathcal{T}$  be the test suites generated at step 4 and step 5 respectively. If  $|\mathcal{T}'| \leq \alpha |\mathcal{T}'_{opt}|$  for some  $\alpha \geq 1$ , then  $|\mathcal{T}| \leq \alpha |\mathcal{T}_{opt}|$ .*

Note that  $|\mathcal{T}_{opt}| = |\mathcal{T}'_{opt}|$  may occur even when  $\mathcal{O}' \not\subseteq \mathcal{O}$ . In our example, the set  $\{I_1, I_2, I_7\}$  lies in  $\mathcal{O}'$  but not in  $\mathcal{O}$ . But the optimal test suites for  $(\mathcal{I}, \mathcal{O})$  and  $(\mathcal{I}', \mathcal{O}')$  both have size eight because (a) at least eight test cases are needed to cover all combinations of  $\{I_1, I_2, I_8\}$ , which lies in both  $\mathcal{O}$  and  $\mathcal{O}'$ , and (b) the test suites we presented for both instances have eight test cases each. On the other hand, it is also possible that  $|\mathcal{T}_{opt}| < |\mathcal{T}'_{opt}|$  when  $\mathcal{O}' \not\subseteq \mathcal{O}$ . We present one such example from Meagher and Stevens [17] in the appendix.

In the next three corollaries, we state additional scenarios when  $|\mathcal{T}_{opt}| = |\mathcal{T}'_{opt}|$ .

**Corollary 3.4** *If the number of colors used in step 1 is  $c$ , and there exists an  $O_i \in \mathcal{O}$  such that  $|O_i| = c$ , then  $|\mathcal{T}_{opt}| = |\mathcal{T}'_{opt}| = l^c$ .*

**Proof.** Since  $O_i \in \mathcal{O}$ , all combinations of  $O_i$  must be covered by any test suite of  $(\mathcal{I}, \mathcal{O})$ . Hence,  $|\mathcal{T}_{opt}| \geq l^{|O_i|} = l^c$ . On the other hand, since  $c$  colors were used in step 1 and  $|O_i| = c$ ,  $\mathcal{I}' = O'_i = \{I'_1, \dots, I'_c\}$ . This means that every other  $O'_j$  is a subset of  $O'_i$ , so  $\mathcal{O}' = \{O'_i\}$  after step 3. Enumerating all the combinations of  $O'_i$  creates an optimal test suite for  $(\mathcal{I}', \mathcal{O}')$ . Thus,  $|\mathcal{T}'_{opt}| = l^c$  and by Theorem 3.2,  $|\mathcal{T}_{opt}| \leq l^c$ . It follows that  $|\mathcal{T}_{opt}| = |\mathcal{T}'_{opt}| = l^c$ .  $\square$

The situation described in Corollary 3.4 typically happens when the size of the largest set in  $\mathcal{O}$  is significantly larger than most of the other sets in  $\mathcal{O}$ . The next result is implied by Corollary 1 of [17]. (We present a proof for completeness.) Recall that for a graph  $G = (V, E)$ ,  $\omega(G)$ , the *clique number* of  $G$ , is the size of the largest  $V' \subseteq V$  such that all vertices in  $V'$  are pairwise adjacent, and  $\chi(G)$ , the *chromatic number* of  $G$ , is the minimum number of colors needed to color  $G$ . It is well known that  $\omega(G) \leq \chi(G)$ .

**Corollary 3.5** *Suppose every set in  $\mathcal{O}$  consists of exactly two parameters. Let  $G$  be the relationship graph of  $(\mathcal{I}, \mathcal{O})$ . If step 1 uses  $\omega(G)$  colors (i.e.,  $\chi(G) = \omega(G)$ ), then  $|\mathcal{T}_{opt}| = |\mathcal{T}'_{opt}|$ .*

**Proof.** Suppose  $\mathcal{I}'' \subseteq \mathcal{I}$  forms a largest clique in  $G$ . Let  $\mathcal{O}''$  consists of all pairs of parameters in  $\mathcal{I}''$ . Since  $\mathcal{O}'' \subseteq \mathcal{O}$ , if  $\mathcal{T}_{opt}$  is an optimal test suite for  $(\mathcal{I}'', \mathcal{O}'')$ , then  $|\mathcal{T}_{opt}| \geq |\mathcal{T}''_{opt}|$ . Since step 1 uses  $\omega(G)$  colors,  $\mathcal{I}' = \{I'_1, \dots, I'_{\omega(G)}\}$ . Different colors were assigned to all the parameters in  $\mathcal{I}''$ . Thus,  $\mathcal{O}'$  consists of all pairs of parameters in  $\mathcal{I}'$ . But  $|\mathcal{I}'| = |\mathcal{I}''|$ , and  $(\mathcal{I}', \mathcal{O}')$  and  $(\mathcal{I}'', \mathcal{O}'')$  have the same structure, so  $|\mathcal{T}'_{opt}| = |\mathcal{T}''_{opt}|$ . And since  $|\mathcal{T}'_{opt}| \geq |\mathcal{T}_{opt}|$  by Theorem 3.2, it follows that  $|\mathcal{T}'_{opt}| = |\mathcal{T}_{opt}|$ .  $\square$

We note that there are many graphs  $G$  where  $\omega(G) = \chi(G)$ , including the family of perfect graphs.

**Corollary 3.6** *If every set in  $\mathcal{O}$  consists of exactly two parameters, and step 1 uses at most three colors, then  $|\mathcal{T}_{opt}| = |\mathcal{T}'_{opt}| = l^2$ .*

**Proof.** Every test suite of  $(\mathcal{I}, \mathcal{O})$  and  $(\mathcal{I}', \mathcal{O}')$  must contain at least  $l^2$  test cases because every output set in  $\mathcal{O}$  and  $\mathcal{O}'$  has  $l^2$  combinations. Hence,  $|\mathcal{T}_{opt}| \geq l^2$  and  $|\mathcal{T}'_{opt}| \geq l^2$ . When step 1 uses two colors,  $\mathcal{I}' = \{I'_1, I'_2\}$  and  $\mathcal{O}' = \{\{I'_1, I'_2\}\}$  so  $\mathcal{T}'_{opt}$  consists of all combinations of  $\{I'_1, I'_2\}$ . Therefore,  $|\mathcal{T}'_{opt}| = l^2$ .

When step 1 uses three colors,  $\mathcal{I}' = \{I'_1, I'_2, I'_3\}$  and  $\mathcal{O}'$  is a subset of  $\mathcal{O}'' = \{\{I'_1, I'_2\}, \{I'_1, I'_3\}, \{I'_2, I'_3\}\}$ . Clearly, every test suite for  $(\mathcal{I}', \mathcal{O}'')$  is also a valid test suite for  $(\mathcal{I}', \mathcal{O}')$ . Construct a  $3 \times l^2$  array whose rows are indexed by  $I'_1, I'_2, I'_3$  as follows. Let the first row consist of  $l$  1's followed by  $l$  2's, and so forth until  $l$   $l$ 's. Let the second row consist of  $l$  sequences of  $1, 2, \dots, l$ , and let the third row start with the sequence of  $1, 2, \dots, l$ , followed by cyclic rotations of this sequence until the sequence  $l, 1, \dots, l-1$  is reached. When  $l = 4$ , the array is as follows:

$I'_1$	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4
$I'_2$	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
$I'_3$	1	2	3	4	2	3	4	1	3	4	1	2	4	1	2	3

It is straightforward to check that every combination of every set in  $\mathcal{O}''$  is covered by one of the columns in the array. Hence,  $(\mathcal{I}', \mathcal{O}'')$  has a test suite of size  $l^2$ , so  $|\mathcal{T}'_{opt}| \leq l^2$  and  $|\mathcal{T}_{opt}| \leq l^2$ .  $\square$

Assuming that every set in  $\mathcal{O}$  consists of exactly two parameters, we can generalize the above corollary as follows. Suppose step 1 uses  $c$  colors, and there is a test suite of size  $l^2$  for  $(\mathcal{I}', \mathcal{O}'')$ , where  $\mathcal{I}' = \{I'_1, \dots, I'_c\}$  and  $\mathcal{O}''$  consist of all pairs of parameters in  $\mathcal{I}'$ . By applying the same reasoning as above, it must be the case that  $|\mathcal{T}_{opt}| = |\mathcal{T}'_{opt}| = l^2$ . In [13], Hartman and Raskin lists the sizes of the smallest test suites that their CTS package has found for various values of  $c$  and  $l$ . From this list, we know for example that when  $c = 4$  and  $l = 3, 4, 5, 7, 8, 9, 11, 12, 13$ , etc. then  $|\mathcal{T}_{opt}| = |\mathcal{T}'_{opt}| = l^2$ .

**Implications and limitations.** We have noted that the main weakness of the greedy algorithm for test generation is that at each iteration it needs to check the coverage of almost all the possible test cases of  $(\mathcal{I}, \mathcal{O})$ . In our case,  $|\mathcal{I}| = k$  and each parameter has  $l$  values each, so there are a total of  $l^k$  test cases. As  $l$  and  $k$  increase, the time and space requirements of the greedy algorithm become prohibitive. The problem reduction approach we introduced tries to make the greedy algorithm more feasible in practice. When say  $|\mathcal{I}'| = k' \ll k$ , then the total number of test cases is reduced to  $l^{k'} \ll l^k$ . In addition, if  $|\mathcal{O}'| < |\mathcal{O}|$ , then the number of combinations that need to be covered is also less. Both the time and space requirements of the algorithm can drop significantly. Unfortunately, it is possible that by applying the problem reduction technique, we will end up with a smaller problem instance whose optimal test suite is larger than the optimal test suite of the original problem instance. Theorem 3.2 and the subsequent corollaries state sufficient conditions as to when our problem reduction technique preserves the optimality of the test suite size.

When is  $|\mathcal{I}'| \ll |\mathcal{I}|$ ? This happens when  $c$ , the number of colors used for  $G$ , is small relative to the number of nodes in  $G$ . In general, this would mean that the relationships between the input parameters based on their occurrences in the  $O_j$ 's are relatively "simple". On the other hand, when every pair of the input parameters occur together in some  $O_j$ , then  $G$  is a complete graph. Consequently,  $|\mathcal{I}'| = |\mathcal{I}|$  and  $|\mathcal{O}'| = |\mathcal{O}|$ , and our technique does not reduce the size of the problem. We note, however, that this is the only instance when no reduction in the number of input parameters is possible. When  $G$  is not a complete graph and  $V$  is its vertex set, there is always a vertex  $v$  whose degree is at most  $|V| - 2$ . The subgraph induced by  $V - \{v\}$  can be colored with  $|V| - 1$  colors while  $v$  can be assigned one of the  $|V| - 1$  colors since it is not adjacent to all of the vertices in  $V - \{v\}$ . Hence,  $\chi(G) \leq |V| - 1$  and  $\mathcal{I}$  can always be reduced by one parameter. Therefore, if we reduce  $(\mathcal{I}, \mathcal{O})$  to  $(\mathcal{I}', \mathcal{O}')$ , and the relationship graph of  $(\mathcal{I}', \mathcal{O}')$  is not a complete graph, then we can again apply our problem reduction technique to  $(\mathcal{I}', \mathcal{O}')$ . We summarize below.

**Remark 3.7** *Suppose the input parameters in  $(\mathcal{I}, \mathcal{O})$  have uniform ranges. Our problem reduction technique can be repeatedly applied to  $(\mathcal{I}, \mathcal{O})$  until the relationship graph of the resulting problem instance is a complete graph.*

Assuming that a reduction in problem size occurs, we will most likely have our greatest gain when  $G$  is colored with as few colors as possible. However, coloring an arbitrary graph is NP-complete [11]. When the size of the graph is not too large, exact algorithms for optimal coloring may be feasible. Recently, Byskov [1] and Eppstein [10] have designed faster algorithms for exact graph coloring. There is also a simple heuristic that colors  $G$  using at most  $\Delta + 1$  colors, where  $\Delta$

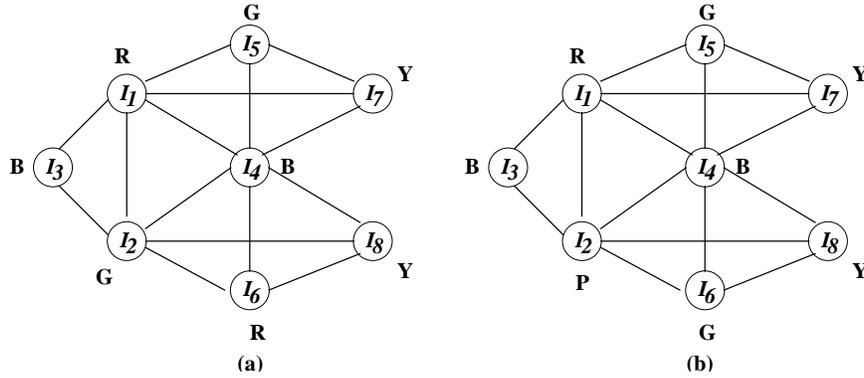


Figure 2: The two graphs are used for generating a test suite for  $\mathcal{I} = \{I_1(5), I_2(5), I_3(5), I_4(5), I_5(6), I_6(6), I_7(2), I_8(2)\}$ , where the numbers in parenthesis indicate the range of each parameter, and  $\mathcal{O} = \{O_1 = \{I_1, I_4, I_5, I_7\}, O_2 = \{I_2, I_4, I_6, I_8\}, O_3 = \{I_1, I_2, I_3\}\}$ . The first graph has a coloring that uses the fewest number of colors. The second graph has a coloring where no two parameters with different range sizes are assigned the same color.

is the maximum degree in  $G$  [23]. Similarly, we know of no algorithm that can efficiently generate an optimal test suite for an arbitrary  $(\mathcal{I}', \mathcal{O}')$ . Hence, we rely on other sub-optimal algorithms such as the greedy algorithm for generating  $\mathcal{T}'$ .

### 3.1 Extensions to parameters with non-uniform ranges

Recall that in our reduction technique, for each color  $s$ , we picked an arbitrary parameter  $I'_s$  whose node in  $G$  was colored  $s$ . In transforming a test case  $T'$  to  $T$ , the values of all parameters whose nodes in  $G$  were colored  $s$  in  $T'$  are then set equal to the value of  $I'_s$  in  $T'$ . If the range of  $I'_s$  is less than the range of one of these parameters, some combinations of  $\mathcal{O}$  may not be covered. We suggest several approaches that deal with this issue, and make use of the following problem instance for illustration:  $\mathcal{I} = \{I_1(5), I_2(5), I_3(5), I_4(5), I_5(6), I_6(6), I_7(2), I_8(2)\}$ , where the numbers in parenthesis indicate the range of each parameter, and  $\mathcal{O} = \{O_1 = \{I_1, I_4, I_5, I_7\}, O_2 = \{I_2, I_4, I_6, I_8\}, O_3 = \{I_1, I_2, I_3\}\}$ .

*Approach 1.* For each color  $s$ , choose  $I'_s$  so that among all the parameters whose nodes in  $G$  are colored  $s$ , its range is the *largest*. The test suite generated by our algorithm is guaranteed to cover all combinations of  $\mathcal{O}$ . The relationship graph for the example is shown in Figure 2(a). We color it with four colors, and choose  $I'_R = I_6$ ,  $I'_G = I_5$ ,  $I'_B = I_4$  and  $I'_Y = I_7$  so  $\mathcal{I}' = \{I_4, I_5, I_6, I_7\}$ . We also have  $O'_1 = O'_2 = \{I_4, I_5, I_6, I_7\}$  and  $O'_3 = \{I_4, I_5, I_6\}$  so  $\mathcal{O}' = \{\{I_4, I_5, I_6, I_7\}\}$ . Generating a test suite for  $(\mathcal{I}', \mathcal{O}')$  is straightforward. The corresponding test suite for  $(\mathcal{I}, \mathcal{O})$  will have  $5 \times 6 \times 6 \times 2 = 360$  test cases.

*Approach 2.* Do not assign two parameters the same color if their ranges are different. One way to implement this is to add an edge in  $G$  between two parameters with unequal ranges. Again, our algorithm will generate a test suite that covers all combinations of  $\mathcal{O}$ . Figure 2(b) shows a coloring of graph  $G$  of the example where only parameters with the same ranges are assigned the same color. This modification forces us to use five colors. We set  $I'_R = I_1$ ,  $I'_B = I_4$ ,  $I'_Y = I_8$ ,  $I'_G = I_5$  and  $I'_P = I_2$  so  $\mathcal{I}' = \{I_1, I_2, I_4, I_5, I_8\}$ . The new output sets are  $O'_1 = \{I_1, I_4, I_5, I_8\}$ ,

$O'_2 = \{I_2, I_4, I_5, I_8\}$ , and  $O'_3 = \{I_1, I_2, I_4\}$ . We have reduced the number of parameters in  $\mathcal{I}$  but not the number of sets in  $\mathcal{O}$ .

*A hybrid approach:* The weakness of the first approach is that we may end up combining parameters with large ranges in the same  $O'_j$  when these parameters were never combined in the original  $O_j$ . Consequently, we may produce a test suite  $\mathcal{T}'$  whose size is significantly larger than that of the optimal test suite for  $(\mathcal{I}, \mathcal{O})$ . To alleviate the problem, we can assign the same color to input parameters with *similar* ranges. For example, add an edge between parameters whose range sizes differ by at least a certain value in the relationship graph, and use the coloring on this graph to construct the reduced problem instance. We note that approach 2 had a stricter rule. This way, we can gain some reduction in problem size, and at the same time prevent  $\mathcal{T}'$  from being too large when compared to the optimal test suite.

## 3.2 Case Studies

The importance of finding efficient techniques for generating test suites for non-uniform instances became apparent in our current research in black-box testing. We have to generate test suites that cover all combinations of input parameters that influence output parameters. The input parameters have non-uniform ranges and the output parameters are often influenced by a different number of input parameters. Currently, we generate test suites using a greedy heuristic similar to the one presented by Cohen, *et al.* [4]. As the systems we experimented with grew in size and complexity, we found that the time and space required to generate these test suites became very large.

Our goal for these case studies was to evaluate the effectiveness of the problem reduction technique in practice. To do so, we implemented the greedy algorithm for test generation on three systems and compared its running time before and after applying the problem reduction technique. We also considered the sizes of the resulting test suites in both instances. Two of the systems we used in the case study are industrial systems that one of the authors was associated with while employed in the software industry. The Digital Access Cross-connect System (DACS) is a hardware and software system used to configure and test digital trunks in telephone systems. The program was written at AT&T Bell Laboratories to provide an easy way to rapidly configure and reconfigure digital trunks in the laboratories where DACS is used in testing telephone switches. The Data Management and Analysis System (DMAS) is used by analytical chemists in the pharmaceutical industry to perform regression analysis on data collected from experiments conducted using liquid and gas chromatography. The third program we used is the Loan Arranger System (LAS), a software used to support the mortgage-back securities business and is described by Pfleeger [18]. LAS is used as a software engineering project at the University of Wisconsin–Milwaukee.

For this study, we determined each system's input-output (I/O) relationships using the automated technique described by Schroeder *et al.* [20]. All programs were executed on a 1.6 Ghz processor with 512 MB of memory. Initially, we ran the greedy algorithm to generate the test suites. The results are shown below. The fourth row indicates the number of input parameters that influence the output parameters. The last row indicates the size of the optimal test suite. For two of the three systems, the greedy algorithm generated the smallest possible test suite.

System	DACS	LAS	DMAS
$ \mathcal{I} $	6	15	23
$ \mathcal{O} $	3	20	42
Sizes of $\mathcal{O}_j$	3	1–6	1–13
Generation time	.39 sec	22.30 sec	870.33 sec
$ \mathcal{T} $	158	1536	5120
$ \mathcal{T}_{opt} $	150	1536	5120

To implement the problem reduction technique, we constructed the relationship graph for each system, used a greedy coloring scheme (p.176 of West [23]) and approach 2 to color the graph. That is, no two parameters' nodes were assigned the same color if their ranges were different. Once  $(\mathcal{I}', \mathcal{O}')$  was constructed, we used the greedy algorithm to generate  $\mathcal{T}'$  at step 4 and the procedure at step 5 to generate the test suite. The results are presented below.

System	DACS	LAS	DMAS
$ \mathcal{I}' $	4	7	13
$ \mathcal{O}' $	3	2	1
Generation time	.05 sec	.66 sec	3.41 sec
$ \mathcal{T}'  =  \mathcal{T} $	162	1536	5120

In each of these systems, the problem reduction technique proved to be an effective way to generate test suites. The generation times are at least one order of magnitude faster, while the resulting test suite sizes are equal or close to that of the optimal test suite.

## 4 Conclusion

In this paper, we addressed the need to generate small test suites efficiently when the combinations that have to be covered are specified by the input-output relationships of a software system. In particular, we did not assume that the input parameters have uniform ranges nor uniform coverage. We revisited the greedy algorithm for test generation and proved that the number of test cases it generates in this setting is at most  $(1 + \ln n) \times OPT$ , where  $n$  is the number of output parameters and  $OPT$  is the size of the optimal test suite. We then presented a problem reduction technique so that when the reduction in problem size is significant, the greedy algorithm and possibly any other test generation method is more efficient in practice. Finally, we discussed its merits and limitations, and evaluated it on several software systems. Some of the questions we would like to pursue further are:

- Suppose  $(\mathcal{I}, \mathcal{O})$  has been reduced to  $(\mathcal{I}', \mathcal{O}')$  by our problem reduction technique, and  $\mathcal{T}_{opt}$  and  $\mathcal{T}'_{opt}$  are their optimal test suites respectively. Does there exist a constant  $\beta$  such that  $|\mathcal{T}'_{opt}| \leq \beta |\mathcal{T}_{opt}|$ ?
- Can the problem reduction technique be modified so that  $|\mathcal{T}_{opt}| = |\mathcal{T}'_{opt}|$  in all cases?

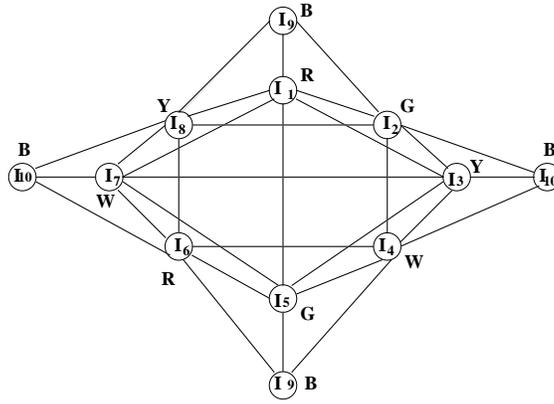


Figure 3: The relationship graph for the problem instance given in the appendix. Nodes  $I_9$  and  $I_{10}$  are drawn twice to make the symmetry of the graph more apparent.

## 5 Appendix

We present an example from Meagher and Stevens [17] for which our problem reduction technique does not preserve the optimality of the test suite size. Consider the following instance:

$\mathcal{I} = \{I_1, I_2, \dots, I_{10}\}$  where all the parameters have  $l = 2$ .

$\mathcal{O} = \{\{I_1, I_2\}, \{I_1, I_3\}, \{I_1, I_5\}, \{I_1, I_7\}, \{I_1, I_8\}, \{I_1, I_9\}, \{I_2, I_3\}, \{I_2, I_4\}, \{I_2, I_8\}, \{I_2, I_9\}, \{I_2, I_{10}\}, \{I_3, I_4\}, \{I_3, I_5\}, \{I_3, I_7\}, \{I_3, I_{10}\}, \{I_4, I_5\}, \{I_4, I_6\}, \{I_4, I_9\}, \{I_4, I_{10}\}, \{I_5, I_6\}, \{I_5, I_7\}, \{I_5, I_9\}, \{I_6, I_7\}, \{I_6, I_8\}, \{I_6, I_9\}, \{I_6, I_{10}\}, \{I_7, I_8\}, \{I_7, I_{10}\}, \{I_8, I_9\}, \{I_8, I_{10}\}\}$

The relationship graph for  $(\mathcal{I}, \mathcal{O})$  is shown in Figure 3. To make the symmetry of the graph more apparent, we followed the suggestion in [17] to repeat nodes  $I_9$  and  $I_{10}$  in the drawing. The graph can be 5-colored, as shown in the figure. Applying steps 2 and 3 of the problem reduction technique, we have  $\mathcal{I}' = \{I_1, I_2, I_3, I_4, I_9\}$  and  $\mathcal{O}'$  consists of all pairs of parameters in  $\mathcal{I}'$ ; i.e., the relationship graph of  $(\mathcal{I}', \mathcal{O}')$  is  $K_5$ , the complete graph on five vertices. It is known that the optimal test suite for  $(\mathcal{I}', \mathcal{O}')$  contains six test cases, so  $|\mathcal{T}'_{opt}| = 6$ . On the other hand, the following is a test suite for the original  $(\mathcal{I}, \mathcal{O})$ .

$I_1$	0	0	1	1	1
$I_2$	0	1	1	0	0
$I_3$	0	1	0	1	1
$I_4$	0	0	1	0	1
$I_5$	0	1	1	1	0
$I_6$	0	0	0	1	1
$I_7$	0	1	1	0	1
$I_8$	0	1	0	1	0
$I_9$	0	1	0	0	1
$I_{10}$	0	0	1	1	0

Hence,  $|\mathcal{T}_{opt}| \leq 5$ , so  $|\mathcal{T}_{opt}| < |\mathcal{T}'_{opt}|$ . If we consider the array formed by rows indexed by  $I_1, I_2, I_3, I_4$  and  $I_9$ , the combination  $(0, 1)$  for  $\{I_1, I_4\}$  and  $\{I_3, I_9\}$  are not covered. We refer the reader to Meagher and Stevens [17] for more examples of problem instances where  $|\mathcal{T}_{opt}| < |\mathcal{T}'_{opt}|$ .

## References

- [1] J. M. Byskov, Algorithms for  $k$ -coloring and finding maximal independent sets, *Proceedings of the Fourteenth ACM-SIAM Symposium on Discrete Algorithms* (2003), 456–457.
- [2] M. Cohen, C. Colbourn, J. Collofello, P. Gibbons and W. Mugridge, Variable strength interaction testing of components, *Proceedings of the 27th Annual International Computer Software and Applications Conference*, (COMPSAC 2003), 413–418.
- [3] M. Cohen, C. Colbourn, P. Gibbons and W. Mugridge, Constructing test suites for interaction testing, *Proceedings of the 25th International Conference on Software Engineering*, (ICSE 2003), 38–48.
- [4] D. M. Cohen, S. R. Dalal, M. L. Fredman and G. C. Patton, The AETG system: an approach to testing based on combinatorial design, *IEEE Transactions on Software Engineering*, 23(7) (2000), 437–444.
- [5] T. Cormen, C. Leiserson and R. Rivest, *Introduction to Algorithms*, MIT Press, 1996, 974–977.
- [6] S.R. Dalal, A. Jain, N. Karunanithi, J.M. Leaton, C. Lott, G. C. Patton and B.M. Harowitz, Model-based testing in practice, *Proceeding of the International Conference on Software Engineering* (1999), 285–294.
- [7] S.R. Dalal and C.L. Mallows, Factor-covering designs for testing software, *Technometrics*, 40(3) (1998), 234–242.
- [8] E. Duesterwald, R. Gupta, and M.L. Soffa, Rigorous data flow testing through output influences, *Proceedings of the Second Irvine Software Symposium* (1992), 131–145.
- [9] I.S. Dunietz, W.K. Ehrlich, B.D. Szablak, C.L. Mallows and A. Iannino, Applying design of experiments to software testing, *Proceedings of the Nineteenth International Conference on Software Engineering* (1997), 205–215.
- [10] D. Eppstein, Small maximal independent sets and faster exact graph coloring, *Proceedings of the Seventh Workshop on Algorithms and Data Structures*, vol. 2125 of *Lecture Notes in Computer Science* (2001), 462–470.
- [11] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.*, W.H. Freeman and Company, 1979.
- [12] A. Hartman, Software and hardware testing using combinatorial covering suites, to appear in *Interdisciplinary Applications of Graph Theory, Combinatorics and Algorithms* (ed. M. Golumbic), manuscript, July 2002.

- [13] A. Hartman and L. Raskin, Problems and algorithms for covering arrays, to appear in the Lindner volume of *Discrete Mathematics*.
- [14] N. Kobayashi, T. Tsuchiya and T. Kikuno, A new method for constructing pair-wise covering designs for software testing, *Information Processing Letters*, 81 (2002), 85–91.
- [15] B. Korel, The program dependence graph in static program testing, *Information Processing Letters* 24(2) (1987), 103–108.
- [16] Y. Lei and K. C. Tai, In-parameter order: a test generation strategy for pairwise testing, *Proceedings of the 3-rd IEEE High-Assurance Systems Engineering Symposium*, 1998, 254–261.
- [17] K. Meagher and B. Stevens, Covering Arrays on Graphs, *submitted to Journal of Combinatorial Theory, Series B*, 2002.
- [18] S.L. Pfleeger, *Software Engineering: Theory and Practice*, Prentice-Hall, 2001.
- [19] Research Triangle Institute, *NIST Planning Report 02-3: The Economic Impacts of Inadequate Infrastructure for Software Testing*”, March 5, 2003, <http://www.nist.gov/director/prog-ofc/report02-3.pdf>.
- [20] P.J. Schroeder, B. Korel, and P. Faherty, Generating expected results for automated black-box testing, *Proceedings of the International Conference on Automated Software Engineering* (2002), 139–48.
- [21] G. Seroussi and N. H. Bshouty, Vector sets for exhaustive testing of digital circuits, *IEEE Transactions on Information Theory*, 34(3) (1988), 513–522.
- [22] N. J. A. Sloane, Covering arrays and intersecting codes, *Journal of Combinatorial Designs*, 1 (1993), 51–63.
- [23] D. West, *Introduction to Graph Theory*, Prentice Hall, 1996, p. 176.