Generating Mixed-Level Covering Arrays with $\lambda = 2$ and Test Prioritization

by

Nicole Ang

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2015 by the
Graduate Supervisory Committee:

Violet Syrotiuk, Chair
Charles Colbourn
Andrea Richa

ARIZONA STATE UNIVERSITY

May 2015

ABSTRACT

In software testing, components are tested individually to make sure each performs as expected. The next step is to confirm that two or more components are able to work together. This stage of testing is often difficult because there can be numerous configurations between just two components.

Covering arrays are one way to ensure a set of tests will cover every possible configuration at least once. However, on systems with many settings, it is computationally intensive to run every possible test. Test prioritization methods can identify tests of greater importance. This concept of test prioritization can help determine which tests can be removed with minimal impact to the overall testing of the system.

This thesis presents three algorithms that generate covering arrays that test the interaction of every two components at least twice. These algorithms extend the functionality of an established greedy test prioritization method to ensure important components are selected in earlier tests. The algorithms are tested on various inputs and the results reveal that on average, the resulting covering arrays are two-fifths to one-half times smaller than a covering array generated through brute force.

ACKNOWLEDGEMENTS

My greatest thanks to Dr. Violet Syrotiuk for patiently reviewing and providing feedback for every step of progress, no matter how small. Your support gave me the confidence to keep moving forward.

I am grateful to my committee members, Dr. Charles Colbourn and Dr. Andrea Richa, for supporting me in this research.

Thank you Dr. Charles Colbourn and Dr. Renée Bryce for developing the greedy test prioritization method. Your algorithm played a large part in optimizing the results of my work.

My sincerest thanks to the staff maintaining the ASU general server. By ensuring the computing resources were always available, I was able to execute numerous tests and produce the results presented in this thesis.

I would like to thank my family and friends for their encouragement and efforts to help me keep a positive mindset.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

A system consists many component or factors, each with its own set of unique settings or values. As systems become more complex, testing procedures to verify the systems' performance also become more complicated. In addition to verifying an individual component works as intended, there is a need to perform tests such that two or more factors work together no matter how they are configured. The reason is because there can be instances where one specific setting in a component can cause an unexpected error in a different piece of the system. As a system grows in complexity, it becomes more difficult to test every possible permutation of factors. The procedure to minimize the number of tests needed to fully test a system is referred to as combinatorial testing [1].

A covering array can be used to sufficiently encompass the main system configurations. The idea is that the covering array acts as a test suite containing a series of tests. Each test contains a set of values that represent a specific setting for each component in the system. In a covering array, λ is used to indicate how many times a specific set of factors is covered. Often, when $\lambda = 1$ meaning every subset of factors is tested at least once, the symbol is omitted [2]. There are numerous methods to construct a covering array, such as heuristics-based, iterative, and artificial life-based [3]. However, there is less research on methods to specifically generate a covering array that covers each interaction at least twice ($\lambda = 2$). The benefit of covering a specific subset of factors twice is that it gives an opportunity to test the subset in conjunction with different settings of the other factors. This may reveal unexpected interactions between components, such as an error.

There are scenarios where there is not enough time or resources to execute every test in the covering array. To combat this issue, test prioritization methods can change the order of tests so that more important components can be tested earlier. An alternative method is change the test generation process to factor in the priorities of certain factors. In this way, a subset of the covering array can be used to test a system. While the subset may not get full coverage, it will cover the features deemed the most important in the system [4].

A simple method of fulfilling $\lambda = 2$ is to first generate a covering array for $\lambda = 1$ by setting each component to its first setting and then iteratively change the values one at a time. A copy of all the tests are then appended to the array. While this method essentially covers every factor combination at least twice, it doubles the size of the test suite. It is possible there is a covering array of a smaller size that can still satisfy $\lambda = 2$. In addition, this test suite does not take into account the priorities of certain factors.

The purpose of this thesis is to develop an algorithm that can generate a mixed level covering array to fulfill the condition of $\lambda = 2$ and use test prioritization to select the tests by order of importance. The fewer tests in the covering array, the better.

One motivation behind this research is that it can reduce the amount of time required to test a system and can more quickly detect errors. This benefit reaches out into the may fields that use covering arrays, such as computer software and hardware[5], biology [6], [7], aviation [8] and finance [9]. Another motivation is that covering arrays can be used in the construction of locating arrays, which focus on finding which factor interactions are causing errors [10]. The smaller the number of tests in a covering array can lead to a smaller locating array, which will improve the efficiency of finding the errors.

Chapter 2 reviews background information and defines terms used throughout the thesis. Chapter 3 describes related work on covering arrays. Chapter 4 explains three algorithms designed in this research. Chapter 5 shows the results of using these algorithms on various inputs. Finally, Chapter 6 summarizes the work accomplished and concludes with a discussion of future work discovered during the implementation of the algorithms.

Chapter 2

BACKGROUND AND DEFINITIONS

Section 2.1 reviews the common terms that are used throughout the thesis. Sections 2.2, 2.3, 2.4, and 2.5 explain nomenclature that use the common terms and provide algorithms and formulas used to generate these components.

2.1 Common Terms

Definition 2.1 *Factor* := $\{f_i\}$

A factor represents a component in a system.

Definition 2.2 *Level* := $\{f_{i,j}\}$

Level j represents a specific value of factor i.

Table 2.1: Example of Factors and Levels for a Book.

		$f_{i,0}$	$f_{i,1}$	$f_{i,2}$
f_0	Type	Electronic	Paperback	Hardcover
f_1	Genre	Non-fiction	Fantasy	Biography
f_2	Language	English	French	Russian

Table 2.1 gives an example of 3 factors, with 3 levels each, for a book. Each row corresponds to a factor, and the columns indicate the levels. An electronic book is denoted by $f_{0,0}$. A book printed in Russian is denoted by $f_{2,2}$.

Definition 2.3 *Test* := $\{f_{0,a}, f_{1,b}, ..., f_{i,j}\}$

A test is a set of factors, where each factor is assigned to a specific level.

For example, using the factors from Table 2.1, $\{f_{0,0}, f_{1,2}, f_{2,1}\}$ is a test for an electronic biography in French.

2.2 Interaction Pair

Definition 2.4 *Interaction Pair* := $\{f_{a,b}, f_{x,y}\}$

An interaction pair represents a combination of two different factors, f_a and f_x, at levels b and y, respectively.

For example, using the factors and levels from Table 2.1, $\{f_{0,1}, f_{2,0}\}$ indicates a paperback book printed in English. A factor is unfixed if it is not set to a level. In this case, the genre of the book is unfixed.

Algorithm 1 Create Interaction Pair Table

```
1:  numOfFactors ← factors.size
2:  Start with an empty table
3:  for i = 0 to numOfFactors − 1 do
4:      for j = i + 1 to numOfFactors − 1 do
5:          Create a column for interaction pairs of f_i and f_j
6:          for k = 0 to factor[i].levels.size do
7:              for l = 0 to factor[j].levels.size do
8:                  Create a new row for the factor-level pair (f_{i,k}, f_{j,l})
9:              end for
10:         end for
11:         Insert the column into the table
12:     end for
13: end for
```

Algorithm 1 generates a table of interaction pairs. Lines 3 and 4 loop through each possible combination of factors. If there are 2 factors, each factor has 2 levels, then the interaction pairs are $\{(f_{0,0}, f_{1,0}), (f_{0,0}, f_{1,1}), (f_{0,1}, f_{1,0}), (f_{0,1}, f_{1,1})\}$. Once all the interaction pairs are created, line 11 inserts the set as a column in the interaction

Table 2.2: Interaction Table of 3 Factors, with 2 Levels Each.

f_0, f_1	f_0, f_2	f_1, f_2
00	00	00
01	01	01
10	10	10
11	11	11

Table 2.3: Interaction Table after Generating Test {0,1,0}.

f_0, f_1	f_0, f_2	f_1, f_2
00	~~00~~	00
~~01~~	01	01
10	10	~~10~~
11	11	11

table. The results of applying this algorithm to 3 factors with 2 levels each is shown in Table 2.2. Each column holds all the interaction pairs for the two factors listed at the top. Column f_0, f_1 lists out all the interaction pairs for factors 0 and 1. Column f_0, f_2 lists out all the interaction pairs for factors 0 and 2. Column f_1, f_2 lists out all the interaction pairs for factors 1 and 2.

As tests are selected for insertion into a covering array under construction, the corresponding interaction pairs are removed from the interaction table. For example, if the test {0,1,0} is generated, Table 2.2 is modified into Table 2.3. {$\underline{0}$,$\underline{1}$,0} is an interaction pair of factors 0 and 1. {$\underline{0}$,1,$\underline{0}$} is an interaction pair of factors 0 and 2. {0,$\underline{1}$,$\underline{0}$} is an interaction pair of factors 1 and 1. The crossed out pairs in Table 2.3 indicate they have been covered and subsequent tests should not contain them, if possible.

2.3 Covering Array

Definition 2.5 *Covering Array* $:= CA_\lambda(N; t, k, v)$

A $N \times k$ array, where N is the total number of rows, k is the total number of factors, t is the strength of coverage in the array, and v is the number of levels for each factor. A t-tuple is a selection of t factors. In every $N \times t$ sub-array, each t-tuple appears at least λ times [11].

Algorithm 2 Naive Covering Array

```
Require: factors[]
 1: function setFactor(i, newTest)
 2:     numOfFactors ← factors.size
 3:     for all levels of factor f_i do
 4:         newTest.factor[i] ← currentlevel
 5:         if f_i is not the last factor then
 6:             return setFactor(i + 1, newTest)
 7:         else
 8:             uncoveredPairs ← 0
 9:             for j = 0 to numOfFactors − 1 do
10:                 for k = j + 1 to numOfFactors do
11:                     if {f_j, f_k} is not covered then
12:                         uncoveredPairs ← uncoveredPairs + 1
13:                     end if
14:                 end for
15:             end for
16:             if uncoveredPairs > 0 then
17:                 Add newTest to test suite
18:                 Update interaction table to mark pairs that are now covered
19:             end if
20:         end if
21:     end for
22: end function
```

Algorithm 2 is a recursive function. Its input is an integer i indicating which factor to set. *newTest* is initialized to a test with unfixed factors and the algorithm iteratively sets the factor to each of its levels. The algorithm uses the information defined in *factors[]* to know how many factors there are and each factors' corresponding levels.

2.3.1 Algorithm 2 Walk-through

The input for this example of the naive covering array algorithm is 3 factors with 2 levels each {*0,1*}. The *2*-tuples for each interaction pair are {(0,0), (0,1), (1,0), (1,1)}.

1. Initialize a test, called newTest, to have unfixed factors, {X,X,X}.

2. Start the recursive algorithm with the call setFactor(*0, newTest*). This call sets the value of factor f_0. Line 3 iterates through each level of factor f_0.

3. Line 4 sets f_0 to level 0. newTest is now {0,X,X}.

4. Line 5 checks if the factor being set is not the last one. For this step, the condition is true since there are 3 factors and only f_0 has been set. Thus, a new call setFactor(*1, newTest*) is executed to set the level of factor f_1.

5. Line 4 sets factor f_1 to level 0. newTest is now {0,0,X}. Since this is not the last factor, another recursive call is executed to set the next factor.

6. Factor f_2 is set to level 0, so newTest becomes {0,0,0}.

7. Since f_2 is the last factor, the algorithm verifies if the test can be selected. Lines 9 - 15 go through each interaction pair of newTest and checks to see if any of the pairs are uncovered, by checking the interaction table shown in Table 2.2. In this iteration, all the pairs, $\{f_{0,0}, f_{1,0}\} = \{0, 0\}$, $\{f_{0,0}, f_{2,0}\} = \{0, 0\}$, $\{f_{1,0}, f_{2,0}\} = \{0, 0\}$, are uncovered.

8. Since newTest covers new interactions previously uncovered, newTest is added to the test suite. The interaction table is updated, as shown in Table 2.4. The interaction pairs in newTest are crossed out, indicating that they are covered.

Table 2.4: Interaction Table after Generating Test {0,0,0}.

f_0, f_1	f_0, f_2	f_1, f_2
~~00~~	~~00~~	~~00~~
01	01	01
10	10	10
11	11	11

9. The algorithm loops back up to line 3. Since it is still in the setFactor() call for factor f_2, line 4 changes newTest to now be {0,0,1}.

10. Lines 9 - 15 find that the interaction pair $\{f_{0,0}, f_{1,0}\} = \{0,0\}$, is covered. However, the pairs $\{f_{0,0}, f_{2,1}\} = \{0, 1\}$ and $\{f_{1,0}, f_{2,1}\} = \{0, 1\}$ are not. Thus, the modified newTest is added to the test suite.

11. The interaction table is updated as shown in Table 2.5.

Table 2.5: Interaction Table after Generating Test {0,0,1}.

f_0, f_1	f_0, f_2	f_1, f_2
~~00~~	~~00~~	~~00~~
01	~~01~~	~~01~~
10	10	10
11	11	11

12. At this point, setFactor() has finished going through every level of factor f_2. It returns to the previous call of setFactor() for factor f_1.

13. Following the action of Step 3, line 4 sets factor f_1 to level 1, which makes newTest be {0,1,X}. A new recursive call is executed to set factor f_2 to each of its levels.

The naive covering array algorithm continues in this manner until it has created a test for every possible combination of the given factors. Table 2.6 shows the resulting test suite.

Table 2.6: Resulting $CA_1(8; 2, 3, 2)$ from Naive Covering Array Walk-through.

Test	f_0	f_1	f_2
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

2.4 Mixed Level Covering Array

Definition 2.6 *Mixed Level Covering Array* := $MCA_\lambda(N; t, k, \{v_1, v_2, \ldots, v_k\})$

A mixed-level covering array "is an $N \times k$ array in which the entries of the ith column arise from an alphabet of size v_i; in addition, choosing any t distinct columns $i_1, ..., i_t$, every t-tuple containing, for $1 \leq j \leq t$, one of the v_{i_j} entries of column i_j, appears in columns $i_1, ..., i_t$ in at least one of the N rows." [12].

Covering arrays are defined such that all factors have the same number of values. However, there are scenarios where some factors may have a different number of levels. To model this type of situation, a mixed level covering array, described in Definition 2.6, is used.

2.4.1 Algorithm 2 Walk-through

Algorithm 2 can be used to generate a mixed-level covering array. The following example is for 3 factors. Factors 0 and 2 have 2 levels {0,1} and factor 1 has only 1 level {0}.

1-11. These steps run in the same way shown in Section 2.3.1 for generating the covering array. The difference is that the interaction is table is different. Therefore in step 8, the algorithm updates the interaction table as shown in Table 2.7.

Table 2.7: Initial Interaction Table in the Mixed-level Covering Array Walk-through.

f_0, f_1	f_0, f_2	f_1, f_2
00	00	00
10	01	01
	10	
	11	

12. At this point, setFactor() has finished going through every level of factor f_2. It returns to the previous call of setFactor() for factor f_1. Since this factor only has one level, the call ends and it returns to setFactor() of f_0. The factor is set to level l_1 and the recursion process begins again.

The naive covering array algorithm continues in this manner until it has created a test for every possible combination of the given factors. Table 2.8 shows the resulting test suite.

Table 2.8: Example of $MCA_1(4; 2, 3, \{2, 1, 2\})$

Test	f_0	f_1	f_2
1	0	0	0
2	0	0	1
3	1	0	0
4	1	0	1

2.5 Weights

Covering arrays produce test suites that cover every possible t-tuple for a set of factors. The naive generation method changes one factor at a time, therefore it

Table 2.9: Example Weight of Each Level for the Input Used in Section 2.4.1.

Factor Level	f_0	f_1	f_2
$f_{i,0}$	0.5	1	0.3
$f_{i,1}$	0.5	-	0.7

includes every possible combination of factors. However, due to time and budget constraints [13], it may not be feasible for all of them to actually be tested on a system. In addition, there may be a need for some tests to be tested earlier because specific factors and levels have a greater importance. Test prioritization methods have been developed to handle these cases by placing tests of greater importance at the beginning of the test suite[4]. These methods can change the order of rows in the covering array, but still give full coverage on all the interaction pairs. One such procedure is to identify specific values of each factor that must be tested earlier and assign them a larger weight[11].

Definition 2.7 *Weight* := $W_{i,j}$

A numerical value that indicates the significance of level j for factor f_i. The sum of the weights of a factor does not need to equal 1.

Table 2.9 gives an example of the input for a mixed level covering array. The weight is split evenly between the two values of factor f_0. The only possible value for factor f_1 is given a weight of 1. For factor f_2, level l_1 has more importance, thus it is given a larger weight. Weights are used in test prioritization methods to determine which test are more significant and need to be tested earlier.

2.5.1 Local Density

Definition 2.8 *Local Density* ($L_{i,j}$) := $W_{i,j} \times U_{i,j}$

Table 2.10: Initial Interaction Table, Local Densities, and Factor Interaction Weights.

Interaction Table		
f_0, f_1	f_0, f_2	f_1, f_2
00	00	00
01	01	01
	10	
	11	

Factor	Local Densities		Total Weight
	$L_{i,0}$	$L_{i,1}$	
f_0	$0.5 \times \frac{3}{3} = 0.5$	$0.5 \times \frac{3}{3} = 0.5$	1
f_1	$1 \times \frac{4}{4} = 1$	-	1
f_2	$0.3 \times \frac{3}{3} = 0.3$	$0.7 \times \frac{3}{3} = 0.7$	1

Local density is a numerical value that indicates the significance of factor f_i's level j. $U_{i,j}$ is the ratio between the number of uncovered interaction pairs and the total number of interaction pairs that factor f_i level j is in.

Before generating any tests, the initial local densities of each factor are calculated, as shown in Table 2.10. After the test, {0,0,0}, is generated, the interaction table is updated and the local densities change to the values shown in Table 2.11. Two out of three interaction pairs containing factor f_0's level 0 are covered. This makes $U_{0,0}$ be $\frac{1}{3}$, which indicates that only 1 interaction pair containing this factor-level is still uncovered. Factor f_0 level 0's local density is changed to $W_{0,0} \times U_{0,0} = 0.5 \times \frac{1}{3} = 0.167$. The local density is less than the original weight because there is less of a need to include this specific level in the next test. Interaction pairs containing factor f_0 level 1 have not been covered yet, therefore $U_{0,1} = \frac{3}{3}$ and the local density is the same as its assigned weight. The local density of every level for each factor is updated in this manner.

2.5.2 Interaction Weight

There are two types of interaction weights, factor interaction weights and factor-level interaction weights.

Table 2.11: Interaction Table, Local Densities, and Factor Interaction Weights after Generating Test {0,0,0}.

Interaction Table		
f_0, f_1	f_0, f_2	f_1, f_2
~~00~~	~~00~~	~~00~~
01	01	01
	10	
	11	

Factor	Local Densities		Total Weight
	$L_{i,0}$	$L_{i,1}$	
f_1	$0.5 \times \frac{1}{3} = 0.167$	$0.5 \times \frac{3}{3} = 0.5$	0.667
f_2	$1 \times \frac{2}{4} = 0.5$	-	0.5
f_3	$0.3 \times \frac{1}{3} = 0.1$	$0.7 \times \frac{3}{3} = 0.7$	0.8

2.5.2.1 Factor Interaction Weight

The factor interaction weight indicates the weight of the uncovered interaction pairs between two factors f_i and f_x. The total weight of a factor is the sum of local densities of its levels. The interaction weight is the product of the total weight of both factors.

Definition 2.9 *Factor Interaction Weight* (I_{f_i,f_x}) := $\sum_{n=0}^{l_i} L_{i,n} \times \sum_{o=0}^{l_x} L_{x,o}$

I_{f_i,f_x} denotes the factor interaction weight between factors f_i and f_x. $\sum_{n=0}^{l_i} L_{i,n}$ is the sum of local densities of each level of factor f_i. $\sum_{o=0}^{l_x} L_{x,o}$ is the sum of local densities of each level for factor f_x.

From the initial local densities in Table 2.10, the initial factor interactions weights are shown in Table 2.12. Each factor is placed in its own row and column. For each cell, where the row and column intersects, the total weight of corresponding factors are multiple together, unless they are same factor. For example, I_{f_0,f_1} is the total weight of factor f_0 multiplied with the total weight of factor f_1. Since no interaction pairs have been covered yet, the total weight of each factor is 1. Thus, all the factor interaction weights are 1.

Table 2.12: Initial Factor Interaction Weights for the Input Used in Section 2.4.1.

	f_0	f_1	f_2
f_0	-	$1 \times 1 = 1$	$1 \times 1 = 1$
f_1	$1 \times 1 = 1$	-	$1 \times 1 = 1$
f_2	$1 \times 1 = 1$	$1 \times 1 = 1$	-

After the test {0,0,0} is generated, the factor interaction weights are adjusted, as shown in Table 2.13. From Table 2.11, factor f_0's total weight is now 0.667, f_1's is 0.5, and f_2's is 0.8. The new factor interaction weights are shown in Table 2.13.

Table 2.13: Factor Interaction Weights after Generating Test {0,0,0}.

	f_0	f_1	f_2
f_0	-	$0.667 \times 0.5 = 0.334$	$0.667 \times 0.8 = 0.534$
f_1	$0.5 \times 0.667 = 0.334$	-	$0.5 \times 0.8 = 0.4$
f_2	$0.8 \times 0.667 = 0.534$	$0.8 \times 0.5 = 0.4$	-

2.5.2.2 Factor-Level Interaction Weight

The factor-level interaction weight is the product of the local densities of factor f_i level j and factor f_x level y. This numerical value indicates the significance of interaction pair $\{f_{i,j}, f_{x,y}\}$.

Definition 2.10 *Factor-Level Interaction Weight* ($I_{f_{i,j},f_{x,y}}$) := $L_{i,j} \times L_{x,y}$

$I_{f_{i,j},f_{x,y}}$ denotes the factor-level interaction weight for the interaction pair of factor f_i level j and factor f_x level y. $L_{i,j}$ is the local density of factor f_i level j and $L_{x,y}$ is the local density of factor f_x level y.

For example, using the weights given in Table 2.9, the factor-level interaction weight $I_{f_{1,1},f_{2,1}} = L_{1,1} \times L_{2,1} = 0.5 \times 1 = 0.5$. When the local densities are updated after test {0,0,0} is selected, shown in Table 2.11, the weight is changed to $I_{f_{1,1},f_{2,1}} = L_{1,1} \times L_{2,1}$ $= 0.167 \times 0.5 = 0.084$.

2.5.3 Weighted Density

Weighted density is the overall significance of an entire test.

Definition 2.11 *Weighted Density of test* := $\frac{\sigma(\{f_{0,a},f_{1,b},...,f_{k,n}\})}{\tau(\{f_0,f_1,...,f_k\})}$

$\sigma(\{f_{0,r}, f_{1,s}, ..., f_{k,t}\}) = \sum_{1\leq a,x\leq k}(c_{f_{a,b},f_{x,y}})$, where

$$c_{f_{a,b},f_{x,y}} = \begin{cases} W_{a,b} \times W_{x,y} & \text{if}\{f_{a,b}, f_{x,y}\} \text{ is not covered} \\ 0 & \text{if}\{f_{a,b}, f_{x,y}\} \text{ is covered} \end{cases}$$

$\sigma()$ calculates the sum of interaction weights for every interaction pair in test $\{f_{0,a}, f_{1,b}, ..., f_{k,n}\}$ to get the total interaction weight of the test. $\tau(\{f_0, f_1, ..., f_k\}) = \sum_{1\leq i,j\leq k} I_{f_i,f_j}$, which calculates the sum of all factor interaction weights.

For example, using the weights defined in Table 2.9, the weighted density of each interaction pair in test {0,0,0} is calculated as follows:

$$\{f_0, f_1\} = \{0, 0\} = I_{f_{0,0},f_{1,0}} = 0.5 \times 1 = 0.5$$
$$\{f_0, f_2\} = \{0, 0\} = I_{f_{0,0},f_{2,0}} = 0.5 \times 0.3 = 0.15$$
$$\{f_1, f_2\} = \{0, 0\} = I_{f_{1,0},f_{2,0}} = 1 \times 0.3 = 0.3$$

The total interaction weight is $0.5 + 0.15 + 0.3 = 0.95$. From Table 2.12, the total factor interaction weight is $I_{f_0,f_1} + I_{f_0,f_2} + I_{f_1,f_2} = 1 + 1 + 1 = 3$. Thus, the weighted density of {0,0,0} is $\frac{0.95}{3} = 0.317$.

Another consideration for calculating weighted density is whether or not the interaction pair has already been covered. If the pair has been covered, there is no added benefit to include it in another test and the interaction weight is set to 0. For example, if the test {0,0,1} is generated after test {0,0,0}, the weighted density of the new test is calculated as follows:

$$\{f_0, f_1\} = \{0, 0\} = 0$$
$$\{f_0, f_2\} = \{0, 1\} = I_{f_{0,0}, f_{2,0}} = 0.5 \times 0.7 = 0.35$$
$$\{f_1, f_2\} = \{0, 1\} = I_{f_{1,0}, f_{2,0}} = 1 \times 0.7 = 0.7$$

The total factor interaction weight remains the same for the entire test suite. Therefore, the weighted density is $\frac{0+0.35+0.7}{3} = 0.35$. Table 2.14 shows the weighted densities of all the tests generated previously.

Table 2.14: Weighted Densities of $MCA_1(4; 2, 3, \{2, 1, 2\})$

Test	f_0	f_1	f_2	Weighted Density
1	0	0	0	0.317
2	0	0	1	0.35
3	1	0	0	0.217
4	1	0	1	0.117

Now that all the important terms used throughout this thesis are defined, Chapter 3 discusses related work.

Chapter 3

RELATED WORK

There are numerous methods that can create a covering array. Grindal et al. surveyed more than 40 papers on strategies to generate test suites and identified 16 categories to define the common methods [3]. Section 3.1, describing test prioritization, falls under the Iterative Test Case Based strategy, which generates one test at a time [3].

3.1 Test Prioritization

Bryce and Colbourn proposed a greedy test prioritization method to generate a mixed-level covering array [11]. The goal was to minimize the size of the covering array for $\lambda = 1$ and select tests in such an order that the most significant tests were at the top of the test suite. The algorithm fixes each factor in the order of descending factor interaction weights. To select a level, the algorithm calculates the weighted density of a test if the factor is fixed to a specific level. The level that produces the highest weighted density is chosen. If there is a tie, then the first occurrence is selected.

$$WeightedDensity := W(f_{a,b}) = \begin{cases} c_{f_{a,b},f_{x,y}} & \text{if factor} f_x \text{ is fixed} \\ \frac{L_{a,b} \times \sum_{y=0}^{l_x} L_{x,y}}{max(\sum_{1 \leq i,j \leq k} I_{f_{i,j}})} & \text{if factor} f_x \text{ is not fixed} \end{cases}$$

This function calculates the weighted density in a slightly different way from Definition 2.11. To account for the fact that some factors are not yet fixed, the piecewise function above shows what is done when calculating each interaction pair if the test uses the factor-level $f_{a,b}$. If factor f_x is fixed, then $c_{f_{a,b},f_{x,y}}$ is used. If factor

f_x is unfixed, then the local density is multiplied by the sum of the local densities of factor f_x. This product is divided by the maximum factor interaction weight.

$$c_{f_{a,b},f_{x,y}} = \begin{cases} L_{a,b} \times L_{x,y} & \text{if}\{f_{a,b}, f_{x,y}\}\text{is not covered} \\ 0 & \text{if}\{f_{a,b}, f_{x,y}\}\text{is covered} \end{cases}$$

Another difference is that $c_{f_{a,b},f_{x,y}}$ uses local densities instead of the initial weights. The reason is that local densities take into account how many interaction pairs have been covered and adjusts the weights to allow other factors to be selected earlier. Using initial weights would make the algorithm select the same level until it has been completely covered before selecting a different level.

When a new test is generated, its interaction pairs are checked off in the interaction table and the local densities of each factor-level is re-calculated. The function is then repeated until all the interaction pairs in the interaction table are covered [11].

Algorithm 3 Greedy Test Prioritization Algorithm [11]

```
1: Initialize an empty test suite
2: while Uncovered pairs remain do
3:     Compute factor interaction weights
4:     Set order of factor selection based on interaction weights
5:     Initialize new test with all factors unfixed
6:     for f = order[0] to order[end] do
7:         Compute factor-level interaction weights of each level of factor f
8:         Select level l for f, which offers the largest increase in weighted density
9:         Fix factor f to level l in new test
10:    end for
11:    Add new test to the test suite
12:    Update interaction table to mark pairs that are covered by test
13:    Update local densities
14: end while
```

3.1.1 Algorithm 3 Walk-through

The following walk-through uses Algorithm 3 to generate the first three tests of a mixed-level covering array for the the input shown in Table 3.1. The values in parentheses are the weights of each factor at each level.

Table 3.1: Input for Algorithm 3 Walk-through

	l_0	l_1	Total Weight
f_0	0 (0.5)	1 (0.5)	1
f_1	0 (1)	-	1
f_2	0 (0.3)	1 (0.7)	1

1. Calculate the factor interaction weights by multiplying the total weight of each factor together. The sum of the weights is the total factor interaction weight of a given level.

	f_0	f_1	f_2	Total I_{f_i}
f_0	-	$1 \times 1 = 1$	$1 \times 1 = 1$	2
f_1	$1 \times 1 = 1$	-	$1 \times 1 = 1$	2
f_2	$1 \times 1 = 1$	$1 \times 1 = 1$	-	2

   All the factor interaction weights are the same, thus the factor selection order, $\{f_0, f_1, f_2\}$, follows the original order. The largest weight ($maxI$) is used to calculate factor-level interaction weights later steps. In this case $maxI$ is 2.

2. Create a new test with unfixed factors, {X,X,X}.

3. In the order determined in Step 1, calculate the weighted density of each level in a factor. Set the new test's factor to the level with the largest weight.

   $W(f_{0,0}) = \frac{L_{0,0} \times \sum_{l=0}^{l_1} L_{1,l}}{maxI} + \frac{L_{0,0} \times \sum_{l=0}^{l_2} L_{2,l}}{maxI} = \frac{0.5 \times 1}{2} + \frac{0.5 \times (0.3+0.7)}{2} = 0.5$

   $W(f_{0,1}) = \frac{L_{0,1} \times \sum_{l=0}^{l_1} L_{1,l}}{maxI} + \frac{L_{0,1} \times \sum_{l=0}^{l_2} L_{2,l}}{maxI} = \frac{0.5 \times 1}{2} + \frac{0.5 \times (0.3+0.7)}{2} = 0.5$

   The weighted densities for both levels of factor f_0 are the same, thus the first level is selected. The test is now {0,X,X}.

   $W(f_{1,0}) = L_{1,0} \times L_{0,0} + \frac{L_{1,0} \times \sum_{l=0}^{l_2} L_{2,l}}{maxI} = (1 \times 0.5) + \frac{1 \times (0.3+0.7)}{2} = 1$

Since factor f_0 is fixed, the interaction weight of $\{f_0, f_1\}$ can be calculated with the weight of factor f_0's assigned level. There is only one level for factor f_1, thus it is selected and the test is now {0,0,X}.

$W(f_{2,0}) = L_{2,0} \times L_{0,0} + L_{2,0} \times L_{1,0} = (0.3 \times 0.5) + (0.3 \times 1) = 0.45$

$W(f_{2,1}) = L_{2,1} \times L_{0,0} + L_{2,1} \times L_{1,0} = (0.7 \times 0.5) + (0.7 \times 1) = 1.05$

A test with $f_{2,1}$ has a higher weighted density than a test with $f_{2,0}$. Therefore, the test is set to {0,0,1}.

4. Add the new test to the test suite. Update the interaction table and local densities, as shown below.

Interaction Table		
f_0, f_1	f_0, f_2	f_1, f_2
~~00~~	00	00
10	~~01~~	~~01~~
	10	
	11	

	$L_{i,0}$	$L_{i,1}$	Total Weight
f_0	$0.5 \times \frac{1}{3} = 0.167$	$0.5 \times \frac{3}{3} = 0.5$	0.667
f_1	$1 \times \frac{2}{4} = 0.5$	-	0.5
f_2	$0.3 \times \frac{3}{3} = 0.3$	$0.7 \times \frac{1}{3} = 0.233$	0.533

5. Calculate the factor interaction weights using the updated local densities.

	f_0	f_1	f_2	Total I_{f_i}
f_0	-	$0.667 \times 0.5 = 0.334$	$0.667 \times 0.533 = 0.356$	0.69
f_1	$0.5 \times 0.667 = 0.334$	-	$0.5 \times 0.533 = 0.267$	0.601
f_2	$0.533 \times 0.667 = 0.356$	$0.533 \times 0.5 = 0.267$	-	0.623

Following the total factor interaction weights in descending order, the factor selection order is set to $\{f_0, f_2, f_1\}$. $maxI$ is set to 0.69.

6. Create a test with unfixed factors {X,X,X}.

7. Calculate the weighted densities of each factor-level in the order determined in Step 5.

$$W(f_{0,0}) = \frac{L_{0,0} \times \sum_{l=0}^{l_1} L_{1,l}}{maxI} + \frac{L_{0,0} \times \sum_{l=0}^{l_2} L_{2,l}}{maxI} = \frac{0.167 \times 0.5}{0.69} + \frac{0.167 \times (0.3+0.233)}{0.69} = 0.25$$

$$W(f_{0,1}) = \frac{L_{0,1} \times \sum_{l=0}^{l_1} L_{1,l}}{maxI} + \frac{L_{0,1} \times \sum_{l=0}^{l_2} L_{2,l}}{maxI} = \frac{0.5 \times 0.5}{0.69} + \frac{0.5 \times (0.3+0.233)}{0.69} = 0.749$$

The weighted density of a test with $f_{0,1}$ is higher, thus the second level is selected. The test is now {1,X,X}. Based on the order chosen in Step 5, factor f_2 is considered next.

$$W(f_{2,0}) = L_{2,0} \times L_{0,0} + \frac{L_{2,0} \times \sum_{l=0}^{l_1} L_{1,l}}{maxI} = (0.3 \times 0.5) + \frac{0.3 \times 0.5}{0.69} = 0.367$$

$$W(f_{2,1}) = L_{2,1} \times L_{0,0} + \frac{L_{2,1} \times \sum_{l=0}^{l_1} L_{1,l}}{maxI} = (0.233 \times 0.5) + \frac{0.233 \times 0.5}{0.69} = 0.285$$

$f_{2,0}$ produces a higher weighted density, thus the test is set to {1,X,0}.

$$W(f_{1,0}) = L_{1,0} \times L_{0,0} + L_{1,0} \times L_{2,0} = (0.5 \times 0.5) + (0.5 \times 0.3) = 0.4$$

Since factors f_0 and f_2 are fixed, multiplication is used to calculate the weighted density of f_1. There is only one level for this factor, so it is selected. The test becomes {1,0,0}.

8. Since all the factors have a level selected, add test {1,0,0} to the test suite. Update the interaction table and local densities, as shown below.

Interaction Table		
f_0, f_1	f_0, f_2	f_1, f_2
~~00~~	00	~~00~~
~~10~~	~~01~~	~~01~~
	~~10~~	
	11	

	$L_{i,0}$	$L_{i,1}$	Total Weight
f_0	$0.5 \times \frac{1}{3} = 0.167$	$0.5 \times \frac{1}{3} = 0.167$	0.334
f_1	$1 \times \frac{0}{4} = 0$	-	0
f_2	$0.3 \times \frac{1}{3} = 0.1$	$0.7 \times \frac{1}{3} = 0.233$	0.333

Repeat Steps 1-4 to generate the third test. This time however, the second aspect of $c_{f_{a,b},f_{x,y}}$ is used when calculating the weighted densities of a possible test.

9. Calculate the factor interaction weights using the updated local densities.

	f_0	f_1	f_2	Total I_{f_i}
f_0	-	$0.334 \times 0 = 0$	$0.334 \times 0.333 = 0.111$	0.111
f_1	$0 \times 0.334 = 0$	-	$0 \times 0.333 = 0$	0
f_2	$0.333 \times 0.334 = 0.111$	$0.334 \times 0 = 0$	-	0.111

The total factor interaction weights of factors f_0 and f_2 are the same, therefore the first factor is chosen first. The factor selection order is set to $\{f_0, f_2, f_1\}$. $maxI$ is set to 0.111.

10. Create a test with unfixed factors, {X,X,X}.

11. In the order determined in Step 9, calculate the weighted density of each level in a factor. Set the test's factor to the level with the largest weight.

$$W(f_{0,0}) = \frac{L_{0,0} \times \sum_{l=0}^{l_1} L_{1,l}}{maxI} + \frac{L_{0,0} \times \sum_{l=0}^{l_2} L_{2,l}}{maxI} = \frac{0.167 \times 0}{0.111} + \frac{0.167 \times (0.1 + 0.233)}{0.111} = 0.501$$

$$W(f_{0,1}) = \frac{L_{0,1} \times \sum_{l=0}^{l_1} L_{1,l}}{maxI} + \frac{L_{0,1} \times \sum_{l=0}^{l_2} L_{2,l}}{maxI} = \frac{0.167 \times 0}{0.111} + \frac{0.167 \times (0.1 + 0.233)}{0.111} = 0.501$$

Both levels produce a test with the same weighted density. The first level is selected and the test becomes {0,X,X}. Based on the order chosen in Step 9, factor f_2 is considered next.

$$W(f_{2,0}) = L_{2,0} \times L_{0,0} + \frac{L_{2,0} \times \sum_{l=0}^{l_1} L_{1,l}}{maxI} = (0.1 \times 0.167) + \frac{0.1 \times 0}{0.111} = 0.017$$

$$W(f_{2,1}) = L_{2,1} \times L_{0,0} + \frac{L_{2,1} \times \sum_{l=0}^{l_1} L_{1,l}}{maxI} = 0 + \frac{0.233 \times 0}{0.111} = 0$$

$f_{2,1}$ has already been paired with $f_{0,0}$, thus the interaction weight is set to 0 because it does not help cover new interaction pairs. This causes level 0 to produce a higher weighted density and the test is set to {0,X,0}.

$$W(f_{1,0}) = L_{1,0} \times L_{0,0} + L_{1,0} \times L_{2,0} = 0 + 0 = 0$$

The interaction pair $\{f_0, f_1\}$ was covered in the first test and $\{f_0, f_1\}$ was covered in the second test. Therefore, the weights of these pairs are set to 0.

However, level l_0 is selected because there is only one level for this factor. The new test becomes {0,0,0}.

12. Add test {0,0,0} to the test suite. Update the interaction table and local densities, as shown below.

Interaction Table		
f_0, f_1	f_0, f_2	f_1, f_2
~~00~~	~~00~~	~~00~~
~~10~~	~~01~~	~~01~~
	~~10~~	
	11	

	$L_{i,0}$	$L_{i,1}$	Total Weight
f_0	$0.5 \times \frac{0}{3} = 0$	$0.5 \times \frac{1}{3} = 0.167$	0.167
f_1	$1 \times \frac{0}{4} = 0$	-	0
f_2	$0.3 \times \frac{0}{3} = 0$	$0.7 \times \frac{1}{3} = 0.233$	0.233

Repeat Steps 1-4 until all the interaction pairs are covered. The resulting mixed-level covering array is shown in Table 3.2.

Table 3.2: Output of Algorithm 3 Walk-through.

Test	f_0	f_1	f_2
1	0	0	1
2	1	0	0
3	0	0	0
4	1	0	1

3.1.1.1 Algorithm 3 Analysis

The tests in Table 3.2 are the same as the tests in Table 2.8, which was generated with the naive covering array algorithm. The difference is the order of the tests. For example, the first test in Table 2.8 is {0,0,0} while Table 3.2 selected test {0,0,1} first. The input in Table 3.1 assigned a larger weight to factor f_2 level 1. This resulted in a higher weighted density for test {0,0,1} than test {0,0,0}, thus the greedy test prioritization method selected {0,0,1} first.

The greedy test prioritization method generates a covering array in which tests are ordered from highest weighted density to lowest. This algorithm is used to generate a covering array to use as a base set of tests for the algorithms discussed in the following chapter.

Chapter 4

COVERING ARRAYS WITH $\lambda = 2$

This chapter presents three algorithms created to generate a covering array that fulfills the condition of $\lambda = 2$. Section 4.1 describes the sample input and base covering array used in each's algorithm walk-through. Section 4.2 discusses the weighted density algorithm. Based on the greedy test prioritization in Section 3.1, this method selects tests based on their weighted densities. This approach ensures that tests with the highest amount of coverage are selected, which reduces the number of tests in the covering array. Section 4.3 explains the general Hamming distance algorithm, which introduces a generalized Hamming Distance value as a metric in selecting a new test. This approach focuses on selecting the most dissimilar tests. Section 4.4 illustrates the weighted Hamming distance algorithm. This design combines the previous two methods to create a covering array with the smallest number of varied tests.

4.1 Algorithm Input

Table 4.1 provides factors and the value and weight (in parentheses) of each level. These values are used as the input for the algorithms discussed in this chapter.

Table 4.1: Values and Weights of the Levels in Each Factor.

	$f_{i,0}$	$f_{i,1}$	$f_{i,2}$	$f_{i,3}$
f_0	0 (.2)	1 (.1)	2 (.1)	3 (.1)
f_1	4 (.2)	5 (.3)	6 (.3)	7 (.2)
f_2	8 (.1)	9 (.7)	10 (.1)	11 (.1)

Algorithm 1 creates a table of all the possible interaction pairs of two factors. Table 4.2 shows the result of creating an interaction table from the sample input.

Table 4.2: Interaction Table for Input for $t = 2$.

f_0, f_1	f_0, f_2	f_1, f_2
0, 4	0, 8	4, 8
0, 5	0, 9	4, 9
0, 6	0, 10	4, 10
0, 7	0, 11	4, 11
1, 4	1, 8	5, 8
1, 5	1, 9	5, 9
1, 6	1, 10	5, 10
1, 7	1, 11	5, 11

f_0, f_1	f_0, f_2	f_1, f_2
2, 4	2, 8	6, 8
2, 5	2, 9	6, 9
2, 6	2, 10	6, 10
2, 7	2, 11	6, 11
3, 4	3, 8	7, 8
3, 5	3, 9	7, 9
3, 6	3, 10	7, 10
3, 7	3, 11	7, 11

The greedy test prioritization method explained in Section 3.1 is used to generate MCA_1. The test suite generated from Algorithm 3 is shown in Table 4.3.

Table 4.3: MCA_1 using Algorithm 3.

Test	f_0	f_1	f_2
1	0	5	9
2	1	6	9
3	2	5	9
4	3	6	9
5	0	4	9
6	0	7	8
7	1	5	10
8	0	6	11
9	2	4	8
10	3	7	9
11	3	5	11

Test	f_0	f_1	f_2
12	2	6	10
13	3	4	10
14	1	7	11
15	1	4	8
16	2	7	8
17	3	5	8
18	0	6	10
19	2	6	11
20	0	6	8
21	0	4	11
22	0	7	10

Once every possible factor-level pair has been covered, $\lambda = 1$ is fulfilled. The next step is to select additional test cases to fulfill $\lambda = 2$. The following algorithms are executed immediately after the greedy test prioritization method. They determine which tests should be added to the existing test suite in Table 4.3. The idea is to

add tests to MCA_1 to make it become a MCA_2. Note, the set of these added tests do not need to create a MCA.

4.2 Weighted Density

Based on the greedy test prioritization method from Section 3.1, another greedy approach is designed to generate tests to fulfill $\lambda = 2$. For example, there are 81 possible tests that can be generated from the input in Table 4.1. The greedy test algorithm selects 22 of them for $\lambda = 1$. The weighted density algorithm first generates the remaining 59 possible combinations of the factors. The weighted density is calculated for each test and the one with the highest value is selected. Algorithm 4 contains the pseudo code for this weighted density (WD) method.

Algorithm 4 Weighted Density Method to Select Tests to Produce MCA_2

```
Require: factors[ ], testSuite[ ]
 1: possibleTests[] ← remaining possible tests
 2: highestDensity ← 0
 3: while Uncovered interaction pairs remain do
 4:     for Each test in possibleTests[ ] do
 5:         weightedDensity ← weighted density of test i
 6:         if weightedDensity > highestDensity then
 7:             Set test i to be the next selected test
 8:             highestDensity ← weightedDensity
 9:         end if
10:     end for
11:     Add indicated test to testSuite and remove it from possibleTests[ ]
12:     Update the interaction table
13: end while
```

4.2.1 Algorithm 4 Walk-through

The following walk-through shows how the weighted density algorithm selects the first two tests to fulfill $\lambda = 2$ given the input in Table 4.1.

1. Generate all possible tests and remove the tests that are already in the test suite. Set all the pairs in the interaction table to be uncovered. The total factor interaction weight used in calculating weighted densities is $0.5 + 1 + 1 = 2.5$.

2. For each possible test, calculate its weighted density. For example, test {0,7,9}'s weighted density is calculated as follows:

$$\{0,7\} = I_{f_{0,0},f_{1,3}} = 0.2 \times 0.2 = 0.04$$
$$\{0,9\} = I_{f_{0,0},f_{2,1}} = 0.2 \times 0.7 = 0.14$$
$$\{7,9\} = I_{f_{1,3},f_{2,1}} = 0.2 \times 0.7 = 0.14$$
$$\text{Weighted Density} = \frac{0.04+0.14+0.14}{2.5} = 0.128$$

3. Add the test with the highest weighted density to the test suite. In the first iteration, the selected test is {0,6,9}. Remove this test from possibleTests[ ].

4. Update the interaction table to indicate that some pairs are now covered by the new test.

f_0, f_1	f_0, f_2	f_1, f_2
0, 4	0, 8	4, 8
0, 5	~~0, 9~~	4, 9
~~0, 6~~	0, 10	4, 10
0, 7	0, 11	4, 11
1, 4	1, 8	5, 8
1, 5	1, 9	5, 9
1, 6	1, 10	5, 10
1, 7	1, 11	5, 11

f_0, f_1	f_0, f_2	f_1, f_2
2, 4	2, 8	6, 8
2, 5	2, 9	~~6, 9~~
2, 6	2, 10	6, 10
2, 7	2, 11	6, 11
3, 4	3, 8	7, 8
3, 5	3, 9	7, 9
3, 6	3, 10	7, 10
3, 7	3, 11	7, 11

Repeat Steps 2-4 to generate the next test.

5. For each possible test, calculate its weighted density. Now that a test has been selected, some of the interaction pairs are covered, which in turn affects the weighted densities for some of the tests. For example, test {0,7,9}'s weighted density is now as follows:

$$\{0,7\} = I_{f_{0,0},f_{1,3}} = 0.2 \times 0.2 = 0.04$$
$$\{0,9\} = 0$$
$$\{7,9\} = I_{f_{1,3},f_{2,1}} = 0.2 \times 0.7 = 0.14$$
$$\text{Weighted Density} = \frac{0.04+0+0.14}{2.5} = 0.072$$

6. Add the test with the highest weighted density to the test suite. In this iteration, the selected test is {1,5,9}. Remove this test from possibleTests[ ].

7. Update the interaction table to indicate some pairs are now covered by the new test.

f_0, f_1	f_0, f_2	f_1, f_2
0, 4	0, 8	4, 8
0, 5	~~0, 9~~	4, 9
~~0, 6~~	0, 10	4, 10
0, 7	0, 11	4, 11
1, 4	1, 8	5, 8
~~1, 5~~	~~1, 9~~	~~5, 9~~
1, 6	1, 10	5, 10
1, 7	1, 11	5, 11

f_0, f_1	f_0, f_2	f_1, f_2
2, 4	2, 8	6, 8
2, 5	2, 9	~~6, 9~~
2, 6	2, 10	6, 10
2, 7	2, 11	6, 11
3, 4	3, 8	7, 8
3, 5	3, 9	7, 9
3, 6	3, 10	7, 10
3, 7	3, 11	7, 11

Repeat Steps 2-4 until all the interaction pairs are covered. All selected tests are appended to the test suite in Table 4.3 to create a covering array that fulfills the condition of $\lambda = 2$. Table 4.4 shows the tests selected using the weighted density (WD) algorithm.

4.3 General Hamming Distance

Producing a covering array of $\lambda = 2$ means there is a great focus on re-testing each t-tuple interaction. This also gives an additional benefit of testing more overall interactions. For example, a covering array contains the test {0,0,0}. The *2*-tuple interaction, (0,0), for factors f_0 and f_1 is tested when factor f_2 is set to 0. The

Table 4.4: Tests added to MCA_1 to make it a MCA_2 using the WD Method.

Test	f_0	f_1	f_2
23	0	6	9
24	1	5	9
25	2	4	9
26	0	7	9
27	0	5	8
28	3	5	9
29	0	4	10
30	1	6	8
31	2	5	10

Test	f_0	f_1	f_2
32	3	6	10
33	1	4	11
34	1	7	10
35	2	7	11
36	3	4	8
37	0	5	11
38	3	7	8
39	2	6	8
40	3	6	11

covering array also has the test {0,0,1}. This covers the same interaction for factors f_0 and f_1 when factor f_2 is set to 1.

An issue with the generated MCA_2 in Table 4.4 is that the selected tests contain a similar combination of factors in MCA_1, shown in Table 4.3. Test 1 in MCA_1 and test 23 in MCA_2 differ by 1 factor. A similar pattern is seen between tests 2 and 24, 3 and 25, as well as most of the other tests. This behavior indicates that the covering array does not effectively test distinct configurations. An idea to ensure tests are more varied is to use a general Hamming distance value.

Definition 4.1 *General Hamming Distance*

$$(T1=\{f_0, f_1, ..., f_n\}, T2=\{f_0, f_1, ..., f_n\}) := \sum_{x=0}^{n} ghd(T1_{f_x}, T2_{f_x})$$

$$ghd(T1_{f_x}, T2f_x) = \begin{cases} 0 & \text{if the levels assigned to } T1_{f_x} \text{ and } T2_{f_x} \text{ are the same} \\ 1 & \text{if the levels assigned to } T1_{f_x} \text{ and } T2_{f_x} \text{ are different} \end{cases}$$

ghd compares factor f_x in two tests, T1 and T2. If both tests assign the same level to factor f_x, *ghd* returns 0. If the levels are different, then *ghd* returns 1.

A Hamming distance value indicates the number of different bits between two binary strings of equal length [14]. It has been used in anti-random testing to achieve

good coverage on the functionality of system [15]. The general Hamming distance value generalizes this idea to indicate the number of factors that are assigned to different levels between two tests. For example, consider the tests T1 = {0,0,0} and T2 = {1,0,1}. Factor f_0 and f_2 are assigned to different levels while factor f_1 is the same in both tests. Since two of the factors are set differently, the general Hamming distance value is 2.

The general Hamming distance algorithm first generates all the tests for every possible combination of factors and ignores the tests that are already selected for MCA_1. The method then calculates the general Hamming distance value and the weighted density of each test. The algorithm selects the test with the highest general Hamming distance value. If there is a tie, the weighted density is used as a tie breaker.

The weighted density is calculated in the same manner used in the weighted density algorithm. The approach to calculate the general Hamming distance is to compare the current set of selected tests with a possible test and store the sum of all the general Hamming distance values. For example, the tests {0,0,0} and {1,0,1} are in a test suite. The test {1,5,8} is a possible test that can be selected. When compared against the first selected test, this possible case has a general Hamming distance value of 3. When compared against the second test, the general Hamming distance value is 2. Therefore, the total general Hamming distance value of {1,5,8} is the sum: $3 + 2 = 5$.

This method selects the tests with the greatest difference from the tests already in the test suite. Algorithm 5 shows the pseudo code of the general Hamming distance (GHD) method.

Algorithm 5 General Hamming Distance Method to Select Tests to Produce MCA_2

Require: factors[ ], testSuite[ ]

```
possibleTests[] ← remaining possible tests
numOfFactors ← factors.size
highestHD ← 0
highestDensity ← 0
while Uncovered interaction pairs remain do
    for Each test i in possibleTests[ ] do
        weightedDensity ← weighted density of test i
        for Each test j in testSuite[ ] do
            for k = 0 to numOfFactors do
                if test_i[k] != test_j[k] then
                    localHD ← localHD + 1
                end if
            end for
        end for
        if localHD > highestHD then
            Set flag to indicate test i should be selected
            highestHD ← localHD
            highestDensity ← weightedDensity
        else if localHD = highestHD then
            if weightedDensity > highestDensity then
                Set flag to indicate test i should be selected
                highestHD ← localHD
                highestDensity ← weightedDensity
            end if
        end if
    end for
    Add the marked test to the test suite. Remove it from possibleTests[ ]
    Update the interaction table
end while
```

4.3.1 Algorithm 5 Walk-through

The following walk-through shows how the general Hamming distance algorithm selects the first two tests to fulfill $\lambda = 2$ given the input in Table 4.1.

1. Generate all possible tests and remove the tests that are already in the test suite produced for $\lambda = 1$. Set all the pairs in the interaction table to be uncovered. The total factor interaction weight used in calculating weighted densities is $0.5 + 1 + 1 = 2.5$.

2. For each possible test:

2a. Calculate its weighted density. For example, test {0,5,11}'s weighted density is calculated as follows:

$$\{0,5\} = I_{f_{0,0},f_{1,1}} = 0.2 \times 0.3 = 0.06$$
$$\{0,11\} = I_{f_{0,0},f_{2,3}} = 0.2 \times 0.1 = 0.02$$
$$\{5,11\} = I_{f_{1,1},f_{2,3}} = 0.3 \times 0.1 = 0.03$$
$$\text{Weighted Density} = \frac{0.06+0.02+0.03}{2.5} = 0.044$$

2b. Calculate the general Hamming distance value. For example, test {0,5,11}'s general Hamming distance value is 48 and is calculated as follows:

Test	f_0	f_1	f_2	GHD
1	0	5	9	1
2	1	6	9	3
3	2	5	9	2
4	3	6	9	3
5	0	4	9	2
6	0	7	8	2
7	1	5	10	2
8	0	6	11	1
9	2	4	8	3
10	3	7	9	3
11	3	5	11	1

Test	f_0	f_1	f_2	GHD
12	2	6	10	3
13	3	4	10	3
14	1	7	11	2
15	1	4	8	3
16	2	7	8	3
17	3	5	8	2
18	0	6	10	2
19	2	6	11	2
20	0	6	8	2
21	0	4	11	1
22	0	7	10	2

3. Add the test with the highest general Hamming distance value to testSuite[ ], using weighted density as a tie-breaker. In the first iteration, the selected test is {1,5,11}. Remove this test from possibleTests[ ].

4. Update the interaction table to indicate some pairs are now covered by the new test.

f_0, f_1	f_0, f_2	f_1, f_2
0, 4	0, 8	4, 8
0, 5	0, 9	4, 9
0, 6	0, 10	4, 10
0, 7	0, 11	4, 11
1, 4	1, 8	5, 8
~~1, 5~~	1, 9	5, 9
1, 6	1, 10	5, 10
1, 7	~~1, 11~~	~~5, 11~~

f_0, f_1	f_0, f_2	f_1, f_2
2, 4	2, 8	6, 8
2, 5	2, 9	6, 9
2, 6	2, 10	6, 10
2, 7	2, 11	6, 11
3, 4	3, 8	7, 8
3, 5	3, 9	7, 9
3, 6	3, 10	7, 10
3, 7	3, 11	7, 11

Repeat Steps 2-4 to select the next test.

5. For each possible test:

5a. Calculate its weighted density. Since a test has been selected, some of the interaction pairs are now covered which, in turn, affects the weighted densities for some of the tests. For example, test {0,5,11}'s weighted density is calculated as follows:

$$\{0,5\} = I_{f_{0,0},f_{1,1}} = 0.2 \times 0.3 = 0.06$$
$$\{0,11\} = I_{f_{0,0},f_{2,3}} = 0.2 \times 0.1 = 0.02$$
$$\{5,11\} = 0$$
$$\text{Weighted Density} = \frac{0.06+0.02+0}{2.5} = 0.032$$

5b. Calculate the general Hamming distance by comparing the test with the test suite, including the test that was added in Step 3. For example, test {0,5,11}'s general Hamming distance value is 49 and is calculated as follows:

Test	f_0	f_1	f_2	GHD
1	0	5	9	1
2	1	6	9	3
3	2	5	9	2
4	3	6	9	3
5	0	4	9	2
6	0	7	8	2
7	1	5	10	2
8	0	6	11	1
9	2	4	8	3
10	3	7	9	3
11	3	5	11	1

Test	f_0	f_1	f_2	GHD
12	2	6	10	3
13	3	4	10	3
14	1	7	11	2
15	1	4	8	3
16	2	7	8	3
17	3	5	8	2
18	0	6	10	2
19	2	6	11	2
20	0	6	8	2
21	0	4	11	1
22	0	7	10	2
23	1	5	11	1

6. Add the test with the highest general Hamming distance value to testSuite[ ], using weighted density as a tie-breaker. In this iteration, the selected test is {1,4,10}. Remove this test from possibleTests[ ].

7. Update the interaction table to indicate some pairs are now covered by the new test.

f_0, f_1	f_0, f_2	f_1, f_2
0, 4	0, 8	4, 8
0, 5	0, 9	4, 9
0, 6	0, 10	~~4, 10~~
0, 7	0, 11	4, 11
~~1, 4~~	1, 8	5, 8
~~1, 5~~	1, 9	5, 9
1, 6	~~1, 10~~	5, 10
1, 7	~~1, 11~~	~~5, 11~~

f_0, f_1	f_0, f_2	f_1, f_2
2, 4	2, 8	6, 8
2, 5	2, 9	6, 9
2, 6	2, 10	6, 10
2, 7	2, 11	6, 11
3, 4	3, 8	7, 8
3, 5	3, 9	7, 9
3, 6	3, 10	7, 10
3, 7	3, 11	7, 11

Repeat Steps 2-4 until all the interaction pairs are covered. Table 4.5 shows the tests that would be added to Table 4.3 to fulfill the requirement $\lambda = 2$.

Table 4.5: Tests added to MCA_1 to make it a MCA_2 using the GHD Method.

Test	f_0	f_1	f_2
23	1	5	11
24	1	4	10
25	2	7	9
26	3	5	10
27	1	7	8
28	2	4	11
29	3	5	9
30	2	6	8
31	3	7	11
32	1	4	9
33	0	5	10
34	1	6	10

Test	f_0	f_1	f_2
35	3	4	8
36	2	7	11
37	0	6	9
38	2	5	8
39	1	7	10
40	3	4	11
41	0	4	8
42	3	6	11
43	2	5	10
44	1	7	9
45	0	7	11

4.4 Weighted Hamming Distance

A problem with the general Hamming distance method is that it often starts with selecting tests that have very low weighted densities because they are the most dissimilar to the currently selected test cases, which have very high weighted densities. This means the values with less significance are tested before values that are deemed more important. The reason for test priority is that if the execution of the tests is halted early, at least the most important features have already been tested. To solve this problem, instead of using weighted densities as a tie-breaker, the idea is to take the general Hamming distance value and multiply it with the weighted density. This combines the characteristics of test diversity (Hamming distance) and factor significance (weights).

The proposed weighted Hamming distance method first generates all the tests for every possible combination of factors and ignore the tests that have already been selected in MCA_1. It calculates the general Hamming distance value and the weighted density of each possible test. Both values are multiplied together. The test with the highest weighted value is selected and added to the test suite. Algorithm 6 shows the pseudo code for the weighted Hamming distance (WHD) method.

Algorithm 6 Weighted Hamming Distance Method to Select Tests to Produce MCA_2

Require: factors[ ], testSuite[ ]
1: $possibleTests[] \leftarrow$ remaining possible tests
2: $numOfFactors \leftarrow factors.size$
3: $highestWeight \leftarrow 0$
4: **while** Uncovered interaction pairs remain **do**
5: **for** Each test in possibleTests[ ] **do**
6: $weightedDensity \leftarrow$ weighted density of test i
7: $localHD \leftarrow$ general Hamming distance value of test i
8: $localWHD \leftarrow weightedDensity \times localHD$
9: **if** $localWHD > highestWeight$ **then**
10: Set flag to indicate test i should be selected
11: $highestWeight \leftarrow localWHD$
12: **end if**
13: **end for**
14: Add the marked test to testSuite[ ]. Remove it from possibleTests[ ]
15: Update the interaction table
16: **end while**
17: Add the indicated test to selectedTests[]

4.4.1 Algorithm 6 Walk-through

The following walk-through shows how the weighted Hamming distance algorithm selects the first two tests to fulfill $\lambda = 2$ given the input in Table 4.1.

1. Generate all possible tests and remove the tests that are already in the test suite. Set all the pairs in the interaction table to be uncovered. The total factor interaction weight used in calculating weighted densities is $0.5 + 1 + 1 = 2.5$.

2. For each possible test:

2a. Calculate its weighted density. For example, test {0,7,9}'s weighted density is calculated as follows:

$$\{0,7\} = 0.2 \times 0.2 = 0.04$$
$$\{0,9\} = 0.2 \times 0.7 = 0.14$$
$$\{7,9\} = 0.2 \times 0.7 = 0.14$$
$$\text{Weighted Density} = \frac{0.04+0.14+0.14}{2.5} = 0.128$$

2b. Calculate its general Hamming distance value. For example, test {0,7,9}'s general hamming distance value is 47 and is calculated as follows:

Test	f_0	f_1	f_2	GHD
1	0	5	9	1
2	1	6	9	2
3	2	5	9	2
4	3	6	9	2
5	0	4	9	1
6	0	7	8	1
7	1	5	10	3
8	0	6	11	2
9	2	4	8	3
10	3	7	9	1
11	3	5	11	3

Test	f_0	f_1	f_2	GHD
12	2	6	10	3
13	3	4	10	3
14	1	7	11	2
15	1	4	8	3
16	2	7	8	2
17	3	5	8	3
18	0	6	10	2
19	2	6	11	3
20	0	6	8	2
21	0	4	11	2
22	0	7	10	1

2c. Multiply together its weighted density and general Hamming distance value. Test {0,7,9}'s weighted Hamming distance value is $0.128 \times 47 = 6.016$.

3. Select the test with the highest weighted Hamming distance value and add it to the test suite. In this iteration, the test selected is {0,6,9}.

4. Update the interaction table to indicate some pairs are now covered by the new test.

f_0, f_1	f_0, f_2	f_1, f_2
0, 4	0, 8	4, 8
0, 5	~~0, 9~~	4, 9
~~0, 6~~	0, 10	4, 10
0, 7	0, 11	4, 11
1, 4	1, 8	5, 8
1, 5	1, 9	5, 9
1, 6	1, 10	5, 10
1, 7	1, 11	5, 11

f_0, f_1	f_0, f_2	f_1, f_2
2, 4	2, 8	6, 8
2, 5	2, 9	~~6, 9~~
2, 6	2, 10	6, 10
2, 7	2, 11	6, 11
3, 4	3, 8	7, 8
3, 5	3, 9	7, 9
3, 6	3, 10	7, 10
3, 7	3, 11	7, 11

Repeat Steps 2-4 to select the next test.

5. For each possible test:

5a. Calculate its weighted density. Some of the interaction pairs are now covered, which causes some tests to have different weighted densities. For example, test {0,7,9}'s weighted density is calculated as follows:

$$\{0,7\} = 0.2 \times 0.2 = 0.04$$
$$\{0,9\} = 0$$
$$\{7,9\} = 0.2 \times 0.7 = 0.14$$
$$\text{Weighted Density} = \frac{0.04+0+0.14}{2.5} = 0.072$$

5b. Calculate its general Hamming distance value. For example, test {0,5,8}'s general Hamming distance value is 48 and is calculated as follows:

Test	f_0	f_1	f_2	GHD
1	0	5	9	1
2	1	6	9	2
3	2	5	9	2
4	3	6	9	2
5	0	4	9	1
6	0	7	8	1
7	1	5	10	3
8	0	6	11	2
9	2	4	8	3
10	3	7	9	1
11	3	5	11	3

Test	f_0	f_1	f_2	GHD
12	2	6	10	3
13	3	4	10	3
14	1	7	11	2
15	1	4	8	3
16	2	7	8	2
17	3	5	8	3
18	0	6	10	2
19	2	6	11	3
20	0	6	8	2
21	0	4	11	2
22	0	7	10	1
23	0	6	9	1

5c. Multiply together its weighted density and general Hamming distance value. Test {0,7,9}'s weighted value is $0.072 \times 48 = 3.456$.

6. Select the test with the highest weighted Hamming distance value and add it to the test suite. In this iteration, the test selected is {0,6,9}.

7. Update the interaction table to indicate that some pairs are now covered by the new test.

f_0, f_1	f_0, f_2	f_1, f_2
0, 4	0, 8	4, 8
0, 5	~~0, 9~~	4, 9
~~0, 6~~	0, 10	4, 10
0, 7	0, 11	4, 11
1, 4	1, 8	5, 8
~~1, 5~~	~~1, 9~~	~~5, 9~~
1, 6	1, 10	5, 10
1, 7	1, 11	5, 11

f_0, f_1	f_0, f_2	f_1, f_2
2, 4	2, 8	6, 8
2, 5	2, 9	~~6, 9~~
2, 6	2, 10	6, 10
2, 7	2, 11	6, 11
3, 4	3, 8	7, 8
3, 5	3, 9	7, 9
3, 6	3, 10	7, 10
3, 7	3, 11	7, 11

Repeat Steps 2-4 until all the interaction pairs are covered. Table 4.6 shows the tests that would be added to Table 4.3 using the weighted Hamming distance (WHD) algorithm.

Table 4.6: Tests added to MCA_1 to make it a MCA_2 using the WHD Method.

Test	f_0	f_1	f_2
23	0	6	9
24	1	5	9
25	2	4	9
26	0	7	9
27	0	5	10
28	3	5	9
29	0	4	8
30	1	6	11
31	2	5	11

Test	f_0	f_1	f_2
32	3	6	10
33	2	6	8
34	1	7	10
35	3	4	11
36	3	7	8
37	1	4	10
38	2	7	11
39	1	5	8
40	0	7	11
41	2	4	10

Three algorithms are proposed in this chapter to generate covering arrays to fulfill the condition of $\lambda = 2$. The weighted density algorithm focuses on selecting tests with the highest amount of interaction coverage. The general Hamming distance algorithm is designed to select tests that provide the most test diversity. The weighted Hamming distance algorithm's goal is to select tests that provide high interaction coverage and test variance. The next chapter examines the algorithms' effectiveness through testing each on a number of factors and levels.

Chapter 5

RESULTS

The greedy test prioritization method in Section 3.1 is used to produce a covering array of $\lambda = 1$. The resulting test suite acts as a base set for the weighted density, general Hamming distance, and weighted Hamming distance algorithms, presented in Chapter 4, to select tests to add onto the covering array and fulfill the condition of $\lambda = 2$. The results of the weighted density, general Hamming distance, and weighted Hamming distance algorithms are labeled "WD", "GHD", and "WHD" respectively.

A naive covering array is included in the results to act as a base comparison to determine the effectiveness of the three algorithms. Algorithm 2 is used to iterate through every possible test and produce a covering array for $\lambda = 1$. All the tests are copied and added to the test suite to fulfill the condition of $\lambda = 2$. The results of this method are labeled as "NCA".

The tables shown in this chapter display the covering array sizes produced from using each algorithm. The MCA_1 column shows the number of tests selected to fulfill the condition of $\lambda = 1$. The MCA_2 column shows the number of added tests selected to fulfill the condition of $\lambda = 2$, i.e. it does not include the number of tests selected previously. The Total column is the total number of tests selected for the MCA_2 covering array.

The figures show the cumulative weight of the covering array for each input. For example, the cumulative weight of test 10 is the sum of the weighted densities of tests 0 - 10. When the cumulative weight reaches 100, all the interaction pairs are covered at least once. When the cumulative weight is 200, the covering array covers all the interaction pairs at least twice. In other words, MCA_1 is complete when the

cumulative weight is 100 and MCA_2 is complete when the cumulative weight is equal to 200.

All the factor levels in the following inputs are assigned a weight of $(\frac{1}{v_{max}})^2$, where v_{max} is the number of levels in the factor [11]. For example, if a factor has 2 levels, then each of its levels are assigned a weight of $(\frac{1}{2})^2 = \frac{1}{4} = 0.25$.

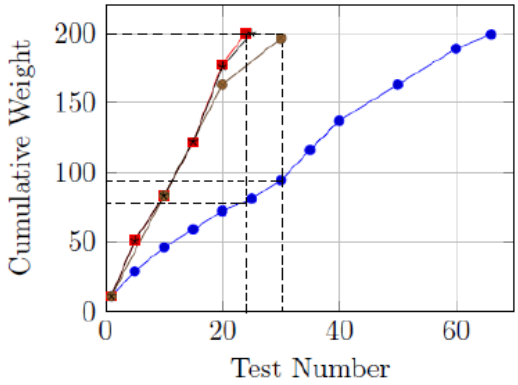
The algorithms attempt to cover every *2-tuple* interactions at least twice. The inputs contain either 3 or 4 factors. As testing progressed, I added more levels to each factor to view the effectiveness of the algorithms as the input size grew. Listed below are the $MCA_2(N; 2, k, \{v_1, v_2, \ldots, v_k\})$ generated when testing the algorithms presented in Chapter 4 in the increasing order of the number of possible tests.

Input 1 Factors f_0, f_1, f_2, f_3 each have 3 levels of weight 0.11. The results are shown in Figure 5.1.

There are 81 possible tests that can be created using Input 1. The weighted density, general Hamming distance, and weighted Hamming distance algorithms completed MCA_2 after adding 11, 19, and 12 tests, respectively, to MCA_1. The Naive covering array algorithm added 33 tests to complete MCA_2. The graph in Figure 5.1 shows that when the presented algorithms reach a cumulative weight of 200, the Naive covering array algorithm has a cumulative weight of around 70 to 90. This means that the Naive covering array algorithm had 70-90% of the interaction pairs at least once when the other algorithms had covered every pair at least twice.

Input 2 Factor f_0 has 10 levels of weight 0.01. Factors f_1, f_2 each have 3 levels of weight 0.11. The results are shown in Figure 5.2.

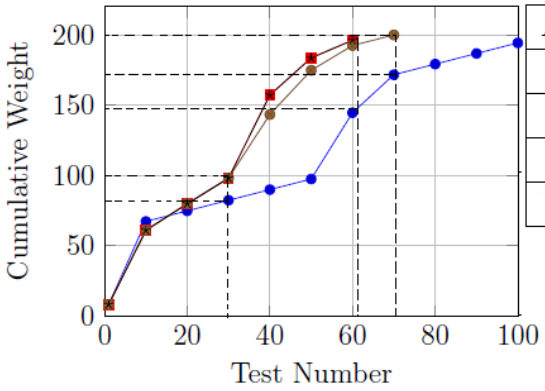
There are 90 possible tests that can be created using Input 2. In this case, the total size of the Naive covering array test suite exceeds this number. Since the



Algorithm	MCA_1	MCA_2	Total
NCA	33	33	66
WD	13	11	24
GHD	13	19	32
WHD	13	12	25

Figure 5.1: Results of Input 1.

algorithm copies the tests of MCA_1 and uses the copy for MCA_2, only 54 of the 90 possible tests are actually used. The weighted density, general Hamming distance, and weighted Hamming distance algorithms reach a cumulative weight of 100 around test 30, while the Naive covering array covers about 80% of the interactions at the same point. When the weighted density and weighted Hamming distance algorithms reached a cumulative weight of 200, the Naive covering array had a cumulative weight about 150. When the general Hamming distance method reached a cumulative weight of 200, the Naive covering array method had a weight of about 170.

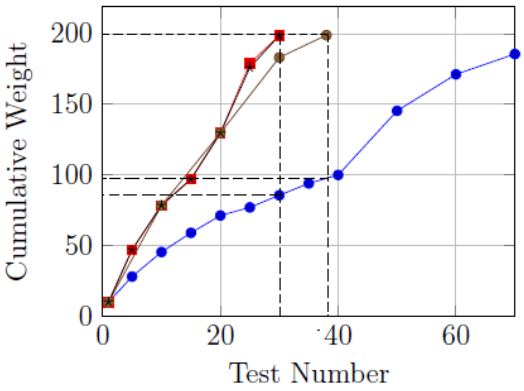


Algorithm	MCA_1	MCA_2	Total
NCA	54	54	108
WD	33	32	65
GHD	33	37	70
WHD	33	31	64

Figure 5.2: Results of Input 2.

Input 3 Factor f_0 has 4 levels of weight 0.0625. Factors f_1, f_2, f_3 each have 3 levels of weight 0.11. The results are shown in Figure 5.3.

There are 108 possible tests that can be created with Input 3. The weighted density, general Hamming distance, and weighted Hamming distance algorithms generate a complete covering array for $\lambda = 2$ in less than 40 tests. The graph in Figure 5.3 shows that when the weighted density and weighted Hamming distance algorithms reach a cumulative weight of 200, the Naive covering array algorithm covered about 80% of the interaction pairs. When the general Hamming distance reaches a cumulative of 200, the Naive covering array is close to 100, meaning it is almost done with covering every interaction pair at least once.



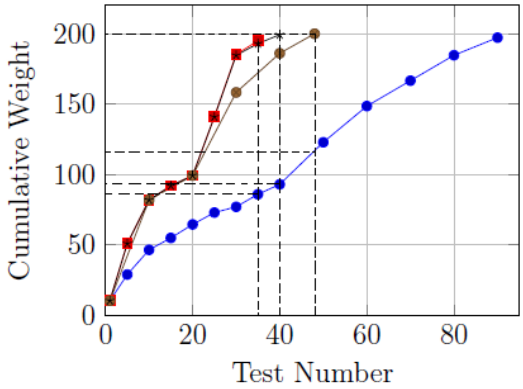
Algorithm	MCA_1	MCA_2	Total
NCA	40	40	80
WD	17	14	31
GHD	17	21	38
WHD	17	14	37

Figure 5.3: Results of Input 3.

Input 4 Factors f_0, f_1, f_2 each have 4 levels of weight 0.0625. Factor f_3 has 2 levels of weight 0.25. The results are shown in Figure 5.4.

There are 128 possible tests that can be selected with Input 4. The weighted density, general Hamming distance, and weighted Hamming distance algorithms add 16, 27, and 20 tests, respectively, to MCA_1 to produce MCA_2. On the other hand, the Naive covering array adds 47 tests to produce MCA_2. When

the presented algorithms reach a cumulative weight of 200, the Naive covering array method has a cumulative between 80 and 120.



Algorithm	MCA_1	MCA_2	Total
NCA	47	47	94
WD	21	16	37
GHD	21	27	48
WHD	21	20	41

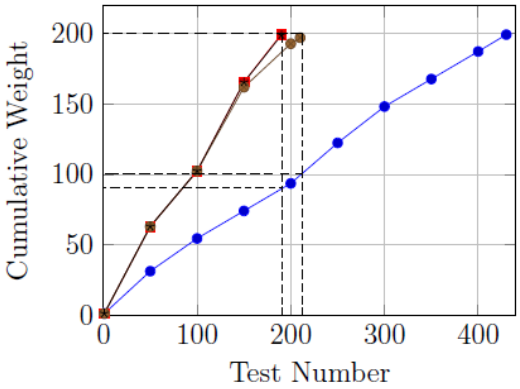
Figure 5.4: Results of Input 4.

The covering array sizes for Input 4 are smaller than the results for Input 3. This is an interesting result since Input 4 has a greater number of possible tests. Intuitively, covering arrays for Input 4 should be larger. The unexpected result occurs because Input 4 has more factors, but fewer levels for each factor. This makes it easier to iterate through all the levels and cover more interaction pairs at a faster rate.

Input 5 Factor f_0 has 10 levels of weight 0.01. Factor f_1 has 9 levels of weight 0.12. Factor f_2 has 8 levels of weight 0.016. The results are shown in Figure 5.5.

There are 720 possible tests that can be selected with Input 4. The three algorithms presented in Chapter 4 produce covering arrays with almost half the number of tests selected by the Naive covering array algorithm. This is shown in the cumulative weight graph. At around 200 tests, the presented algorithms reach a weight of 200 while the Naive covering array method has a cumulative weight of about 100. The naive algorithm selects a total of 432 tests to cover every interaction pair at least twice. The weighted density algorithm selects a

total of 191 tests, the general Hamming distance algorithm selects a total of 216 tests, and the weighted Hamming distance algorithm selects a total of 193 tests.



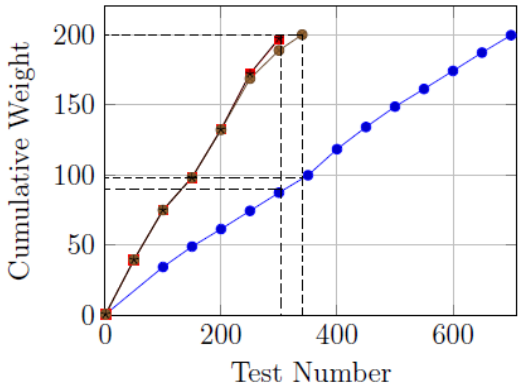
Algorithm	MCA_1	MCA_2	Total
NCA	216	216	432
WD	98	93	191
GHD	98	118	216
WHD	98	95	193

Figure 5.5: Results of Input 5.

Input 6 Factors f_0, f_1 each have 12 levels of weight 0.0069. Factor f_2 has 10 levels of weight 0.01. The results are shown in Figure 5.6.

There are 1440 possible tests that can be generated from Input 6. The Naive covering array algorithm produces a covering array of size 702 to cover every interaction pair at least twice. The weighted density algorithm selects a total of 316 tests, the general Hamming distance algorithm selects a total of 340 tests, and the weighted Hamming distance algorithm selects a total of 313 tests. The graph in Figure 5.6 shows that the three presented algorithms reach a cumulative weight of 200 when the Naive covering array method has a cumulative weight of 90 to 100.

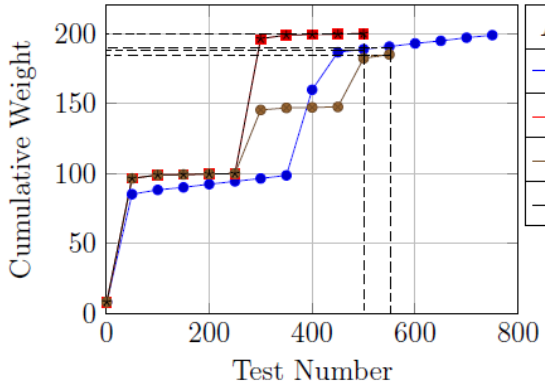
Input 7 Factor f_0 has 25 levels of weight 0.0016. Factor f_1 has 10 levels of weight 0.01. Factor f_2 has 4 levels of weight 0.25. Factor f_3 has 2 levels of weight 0.25. The results are shown in Figure 5.7.



Algorithm	MCA_1	MCA_2	Total
NCA	351	351	702
WD	159	157	316
GHD	159	181	340
WHD	159	154	313

Figure 5.6: Results of Input 6.

There are 2000 possible tests for Input 7. The Naive covering array algorithm generates a covering array for $\lambda = 2$ with 778 tests. The weighted density and weighted Hamming distance algorithms both select a total of 501 tests. The general Hamming distance algorithm selects 563 tests.



Algorithm	MCA_1	MCA_2	Total
NCA	389	389	778
WD	251	250	501
GHD	251	312	563
WHD	251	250	501

Figure 5.7: Results of Input 7.

In the previous input cases, the cumulative weight of the general Hamming distance algorithm increased at a similar rate to the weighted density and weighted Hamming distance algorithm. For Input 7, the general Hamming distance algorithm's cumulative weight increases at about half the rate as the other two algorithms. This indicates that the tests selected are covering fewer interaction pairs, which drives the algorithm to select more tests to get full coverage.

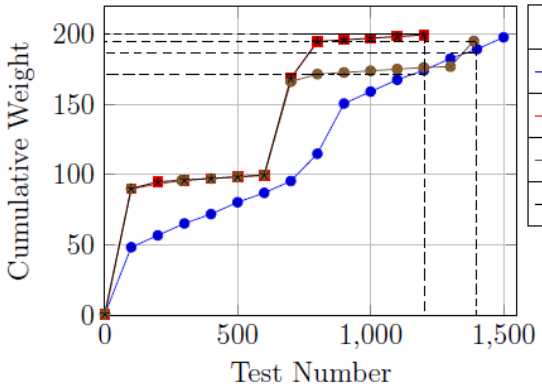
The tests cover so few interactions, that after about 550 tests, the general Hamming distance method had a smaller cumulative weight the Naive covering array method.

Another interesting result is that the weighted density and weighted Hamming distance algorithms produced covering arrays that are less than half the size of covering array generated by the naive algorithm. The result of this test input breaks the pattern of completing with half the number of tests as seen in the previous input cases.

Input 8 Factors f_0, f_1 each have 25 levels of weight 0.0016. Factor f_2 has 4 levels of weight 0.0625. The results are shown in Figure 5.8.

There are 2500 possible tests that can be selected for Input 8. The Naive covering array algorithm generates a covering array with 1544 tests. The weighted density algorithm selects a total of 1252 tests. The weighted Hamming distance algorithm selects 1253 tests. The general Hamming distance algorithm selects 1388 tests. In Figure 5.8's graph, the weighted density and weighted Hamming distance algorithms reach cumulative weight of 200, while the Naive covering array algorithms has a weight of about 165 at this point. In the graph, at around 1400 tests, the cumulative weights of general Hamming distance and Naive covering array algorithms are almost the same.

Similar to the scenario in Input 7, the general Hamming distance algorithm selects tests that covers very few interaction pairs. These selections force the algorithm to continue adding more tests to reach full coverage. Another interesting result is that all three algorithms presented in Chapter 4 generated covering arrays with nearly the same number of tests generated by the Naive covering array algorithm. However, the smallest possible of tests to cover every interaction



Algorithm	MCA_1	MCA_2	Total
NCA	772	772	1544
WD	627	625	1252
GHD	627	761	1388
WHD	627	626	1253

Figure 5.8: Results of Input 8.

is the product between the factors with the most levels, $25 \times 25 = 625$. This means the lower bound on the covering array size for $\lambda = 2$ is $625 \times 2 = 1250$. Since, the algorithms produce covering arrays close to this lower bound, they are still efficient despite being similar in size to the naive method.

5.1 Results Analysis

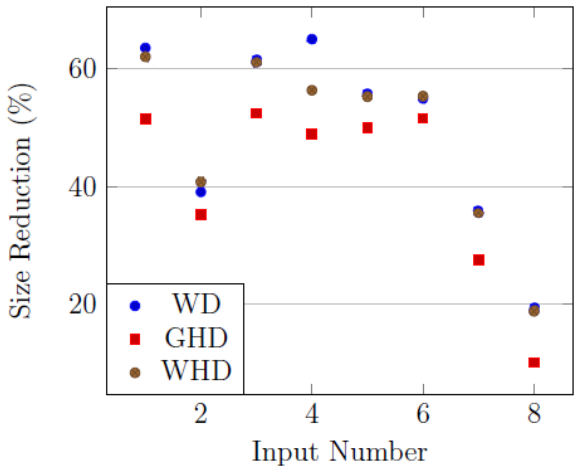


Figure 5.9: Size Ratios between Algorithms.

Figure 5.9 displays the size ratio of the algorithms for each input. The size ratio is the number of tests selected by each algorithm presented in Chapter 4 divided by the number of tests selected by the naive covering array algorithm. For example,

the size ratio of weighted density to Naive CA for input 1 is $100 - (\frac{13+11}{33+33} \times 100) = 100 - (\frac{24}{66} \times 100) = 100 - (0.364 \times 100) = 100 - 36.4 = 63.6$. Therefore, the weighted density test suite is 36.4% the size of the Naive CA test suite.

On average, the weighted density algorithm produces a test suite that is 49.3% smaller than the Naive CA test suite. The general Hamming distance test suite is on average 40.9% smaller than the Naive CA test suite. The weighted Hamming distance algorithm produced test suites that are on average 48.2% smaller than the Naive CA test suite.

The general Hamming distance algorithm consistently produced larger test suites than the other two algorithms. As the input gained more factors and levels, the number of tests selected increased until it was almost equal in size to the test suite generated by the naive covering array algorithm. The reason for this is that the algorithm neither cares about the weights of the factor-levels nor how many interaction pairs a test may cover. This causes the algorithm to skip over tests that provide more coverage in favor of a test that is different from the already assigned tests.

The ineffectiveness in covering interaction pairs is expected in the general Hamming distance algorithm. The main goal of its design is to select tests of greater variety.

Figure 5.10 shows a 3-dimensional representation of the covering arrays generated by the Naive covering array and general Hamming distance methods for Input 6. The figure uses dots to indicate tests selected for MCA_1 and triangles for tests selected for MCA_2. The Naive covering array algorithm changes factor one at a time in a sequential order, which leads to the diagram containing tests that appear along the border axes. The tests selected by the general Hamming distance algorithm appear relatively scattered all around the 3-dimensional space. This indicates the tests are

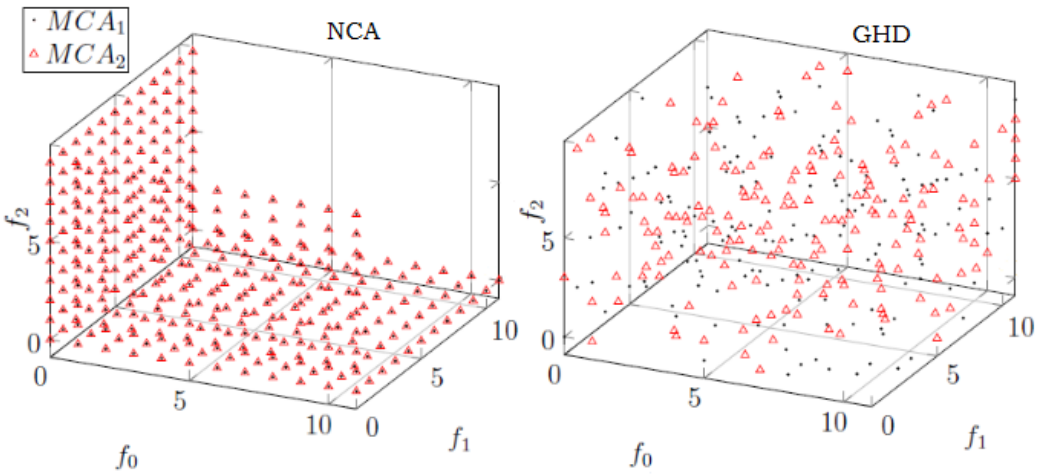


Figure 5.10: 3D Representation of NCA and GHD Arrays from Input 6.

varied enough to ensure specific factor levels are tested with different levels of the other factors.

Figure 5.11 shows a 3-dimensional representation of the covering arrays generated by the general Hamming distance method for Input 6 and 8. The figure uses dots to indicate tests selected for MCA_1 and triangles for tests selected for MCA_2.

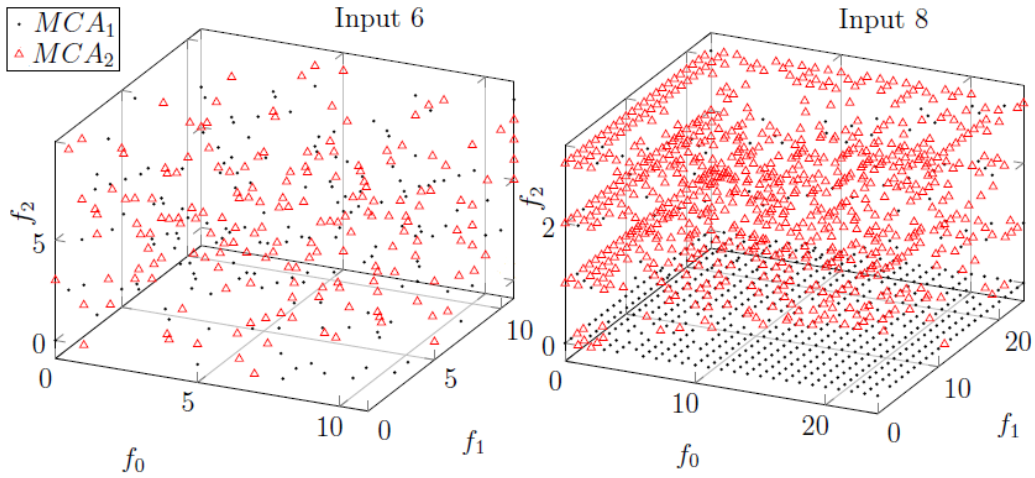


Figure 5.11: 3D Representation of GHD Arrays from Input 6 and 8.

Similar to the diagram of Input 6, the 3-dimensional graph for Input 8 shows that the tests are randomly scattered. The difference is that some of the points form a line along the axes' boundaries. This happens because the general Hamming distance

algorithm selects so many tests in the middle of the 3-dimensional space that it starts to select the border tests in almost sequential order. For Input 8, the last 100 tests closely followed the pattern of {0,0,2}, {0,1,1},...,{0,24,1},{1,0,0}, and so on. This situation can be avoided by selecting tests that also consider weight.

The weighted density and weighted Hamming distance algorithms produced covering arrays that are of similar size and have a similar increase in cumulative weight for all the inputs tested. Inspection of the actual tests selected revealed the differences between these two algorithms.

The 3-dimensional representation in Figure 5.12 shows the weighted Hamming distance algorithm selects tests similar to the weighted density algorithm for Input 6. The figure uses dots to indicate the tests selected for MCA_1 and triangles for tests selected for MCA_2.

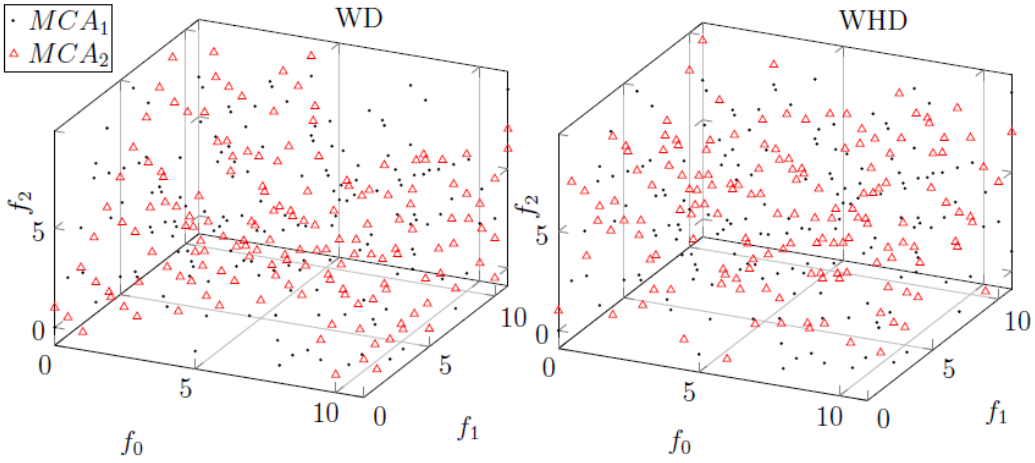


Figure 5.12: 3D Representation of WD and WHD Arrays from Input 6.

However, as more factors and levels are added to the input, the weighted Hamming distance algorithm begins to diverge because it starts to select tests that have the same weight, but a greater Hamming distance value. Since the generalized Hamming distance value does not take into account interaction pair coverage, it can drive the algorithm to select a test that covers less interaction pairs. This is the main reason

that the weighted Hamming distance algorithm produces a slightly larger covering array for most of the input tested in this chapter.

While the weighted Hamming distance algorithm does not show any improvement in terms of size, it does address the need of generating dissimilar tests. The ability to ensure this condition became more apparent as the input size increased. Figure 5.13 shows a 3-dimensional representation of the covering arrays generated by the weighted density and weighted Hamming distance algorithms for Input 8. The figure uses dots to indicate the tests selected for MCA_1 and triangles for tests selected for MCA_2.

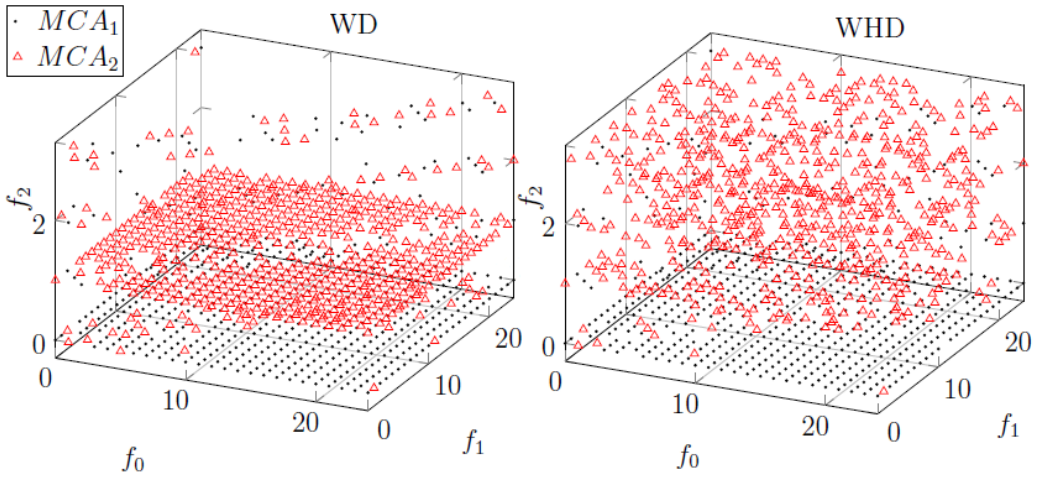


Figure 5.13: 3D Representation of WD and WHD Arrays from Input 8.

Once all the interaction pairs of a factor are covered, the weighted density algorithm typically sets that factor to level 0 on subsequent tests in MCA_1. During the selection process of MCA_2, fully covered factors are typically assigned to level 1. This behavior causes the scatter plot to almost fill an entire plane for the first and second levels of factor f_2. The scatter plot also shows that MCA_1 and MCA_2 contain a cluster of tests in the form of a diagonal line when factor $f_2 = 2$ or $f_2 = 3$. This indicates that factors f_0 and f_1 are assigned to similar levels when factor f_2 is set to level 2 or 3.

On the other hand, the weighted Hamming distance algorithm selects the tests for MCA_2 appear randomly in its scatter plot. This reveals that tests are not selected in a predictable manner. It also indicates that each factor-level is included in multiple tests regardless of whether or not it has been fully covered. This means the tests get the extra benefit of testing a specific factor in conjunction with different settings.

5.2 Summary

All three algorithms presented in Chapter 4 can produce covering arrays that cover every interaction pair at least twice. The general Hamming distance and weighted Hamming distance algorithms produce covering arrays with the extra benefit of testing a specific factor in various settings. In terms of size, all the algorithms select fewer tests than an array generated by a naive covering array algorithm. In terms of test prioritization, the weighted density and weighted Hamming distance algorithms fulfill this requirement for the entire test suite. The general Hamming distance algorithm does not satisfy the test prioritization requirement for MCA_2.

Chapter 6

CONCLUSIONS & FUTURE WORK

The purpose of this thesis is to generate covering arrays to cover every interaction pair at least twice. All three presented algorithms can produce covering arrays with full coverage for $\lambda = 2$ without having to generate every possible test.

The weighted density and weighted Hamming distance algorithms consider the weights of all the factors and levels and are able to select tests in an order of significance. While both algorithms generate smaller covering arrays than a naive covering array, the weighted density algorithm performs better in terms of size.

The general Hamming distance and weighted Hamming distance algorithms generate covering arrays that provide test diversity. The former algorithm does not take into account interaction pair coverage, thus its selection process becomes less effective as the input becomes larger. The weighted Hamming distance algorithm is the better option to construct a covering array with test diversity.

Overall, the weighted Hamming distance algorithm most effectively produces a prioritized mixed-level covering array that covers every interaction pair twice. The generated test suites of this method are not the smallest, but it provides the test diversity desired when testing systems with multiple components.

6.1 Future Work

This research focused on getting coverage for interaction pairs that represent a combination of two different factors, also known as *2-wise* or 100% *pair-wise* coverage. A possible direction for future work is to automatically produce covering arrays for *t-wise* coverage, where $t \geq 3$. This can be further extended by considering *variable*

strength coverage. In this case, subsets of the factors require different *t-wise* coverage [3].

The algorithms presented in this thesis generated all the possible tests and examined each one. The drawback to this idea is that as the input becomes larger, the number of possible tests can grow exponentially and storing the set will use too many resources. This implementation approach was used because the test input were relatively small, thus the programs would quickly iterate through the set of possible tests. However this method cannot be used in real-world applications where there are multiple factors and levels. A different implementation idea is to create the next test by setting the factors one-by-one, similar to the greedy test prioritization algorithm. This approach reduces the amount of space used since it only stores the tests in the covering array. The reason this method was not used in this thesis was because it is more computational intensive and causes the execution time to be longer.

Another consideration for future work is to handle different weights for each level, as seen in the tests using the greedy test prioritization algorithm [11]. Run-time and memory allocation errors sometimes occurred when testing the presented algorithms on various inputs. The main cause was how each levels' weight was set. Different weight values for a factor are not used in Chapter 5 (i.e., level 0's weight is 0.3 and level 1's weight is 0.9). The reason is that a level with a large weight retains a high local density until most of its interaction pairs are covered. For example, a level with a weight of 0.9 needs to have $\frac{8}{9}$ of its interaction pairs covered to get a local density of 0.1. Until this occurs, the algorithms skips over levels with weights of 0.1. Once this implementation issue is solved, the input factors and levels used in Chapter 5 can be assigned different weights and the results should produce very different covering arrays.

There are instances where one specific setting in one component causes an error when working in conjunction with another piece. If these cases are known, tests containing the undesired interaction pairs should not be selected. Nie and Leung note that some work has been done to address constraint factors to indicate invalid values [4].

In the opposite scenario, a certain test must be included in the test suite. A possible direction to implement this functionality is to use seeding. Nie and Leung define seeding as a "means to assign some specific test cases ... in testing"[4].

Covering arrays are a strategy often used in testing large systems. The work accomplished here is another step in the ongoing process to improve the efficiency of selecting fewer tests with higher coverage.

REFERENCES

[1] R. Kuhn, *Introduction to combinatorial testing*, Jun. 2011. [Online]. Available: `http://mse.isri.cmu.edu/software-engineering/documents/faculty-publications/miranda/kuhnintroductioncombinatorialtesting.pdf`.

[2] V. V. Kuliamin and A. Petukhov, "A survey of methods for constructing covering arrays", *Programming and Computer Software*, vol. 37, no. 3, pp. 121–146, 2011.

[3] M. Grindal, J. Offutt, and S. F. Andler, "Combination testing strategies: a survey", *Software Testing, Verification and Reliability*, vol. 15, no. 3, pp. 167–199, 2005.

[4] C. Nie and H. Leung, "A survey of combinatorial testing", *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, p. 11, 2011.

[5] A. Hartman, "Software and hardware testing using combinatorial covering suites", in *Graph Theory, Combinatorics and Algorithms*, Springer, 2005, pp. 237–266.

[6] A. H. Ronneseth and C. J. Colbourn, "Merging covering arrays and compressing multiple sequence alignments", *Discrete Applied Mathematics*, vol. 157, no. 9, pp. 2177–2190, 2009.

[7] D. E. Shasha, A. Y. Kouranov, L. V. Lejay, M. F. Chou, and G. M. Coruzzi, "Using combinatorial design to study regulation by multiple input signals. a tool for parsimony in the post-genomics era", *Plant Physiology*, vol. 127, no. 4, pp. 1590–1594, 2001.

[8] R. Bartholomew, "An industry proof-of-concept demonstration of automated combinatorial test", in *Automation of Software Test (AST), 2013 8th International Workshop on*, IEEE, 2013, pp. 118–124.

[9] M. Mehta and R. Philip, "Applications of combinatorial testing methods for breakthrough results in software testing", in *Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on*, IEEE, 2013, pp. 348–351.

[10] C. Martínez, L. Moura, D. Panario, and B. Stevens, "Algorithms to locate errors using covering arrays", English, in *LATIN 2008: Theoretical Informatics*, ser. Lecture Notes in Computer Science, vol. 4957, Springer Berlin Heidelberg, 2008, pp. 504–519.

[11] R. C. Bryce and C. J. Colbourn, "Test prioritization for pairwise interaction coverage", *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, May 2005.

[12] C. J. Colbourn, S. S. Martirosyan, G. L. Mullen, D. Shasha, G. B. Sherwood, and J. L. Yucas, "Products of mixed covering arrays of strength two", *Journal of Combinatorial Designs*, vol. 14, no. 2, pp. 124–138, 2006.

[13] A. Hartman and L. Raskin, "Problems and algorithms for covering arrays", *Discrete Mathematics*, vol. 284, no. 1, pp. 149–156, 2004.

[14] R. C. Bryce, C. J. Colbourn, and D. R. Kuhn, "Finding interaction faults adaptively using distance-based strategies", in *Engineering of Computer Based Systems (ECBS), 2011 18th IEEE International Conference and Workshops on*, IEEE, 2011, pp. 4–13.

[15] S. Wu, Y. K. Malaiya, and A. P. Jayasumana, "Antirandom vs. pseudorandom testing", in *Computer Design: VLSI in Computers and Processors, 1998. ICCD'98. Proceedings. International Conference on*, IEEE, 1998, pp. 221–223.