Constructions of Covering Arrays

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Reshma C Chandrasekharan

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April, 2015

Supervisor: Dr. Soumen Maity

This is to certify that this dissertation entitled Constructions of Covering Arrays towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents the research carried out by Reshma C Chandrasekharan at Indian Institute of Science Education and Research under the supervision of Dr. Soumen Maity, Associate Professor, Department of Mathematics, during the academic year 2014-2015.

[illegible]

Dr. Soumen Maity

Committee:
Dr. Soumen Maity
Dr. Rabeya Basu

Amma, Achan, Kannan, Madhu and Soumen sir.

Declaration

I hereby declare that the matter embodied in the report entitled Constructions of Covering Arrays are the results of the investigations carried out by me at the Department of Mathematics, Indian Institute of Science Education and Research Pune, under the supervision of Dr Soumen Maity and the same has not been submitted elsewhere for any other degree.

[illegible]

Reshma C Chandrasekharan

Acknowledgments

I take this opportunity to express my deepest gratitude to my guide Dr. Soumen Maity for the constant guidance, inspiration and continuous support. It has been my privilege to work with him and I am indebted to him for his patience and all that I have learned being his student. I thank Dr. Rabeya Basu and Dr. Anupam Kumar Singh for their exceptional care and concern towards us. I extend my gratitude to all my teachers.

I would like to thank IISER Pune for giving us the opportunity to participate in research at an early stage and for providing the necessary computational facilities.

Without the strong support and love from my parents and my brother, this project would not have happened. I thank Madhu, Jithin, Sukruti and Vishnu for being with me during the difficult times. Lastly, I thank all my friends and relatives for their support.

Abstract

A covering array of size n, strength t, degree k and order g is a $k \times n$ array on a set of g symbols with the property that in each $t \times n$ subarray, every $t \times 1$ column appears at least once. Covering arrays have been studied for their applications in the testing of software, hardware, network etc. It is desirable in most applications to minimize the size n of a covering array. In this thesis, we propose techniques for constructing good covering arrays using group theory coupled with computer search. In 2004, Meagher and Stevens developed group construction of covering arrays of strength two which uses an array and a group action on the array. This method employs the action on the symbols of a group of order $g - 1$ fixing one symbol. We extend this method so that the number of fixed symbols is permitted to take any non-negative integer value. A comparison of our method with heuristic tools like NIST IPOG-F shows that our construction produces significantly smaller size covering arrays. We also propose a technique for constructing covering arrays of strength three with budget constraints.

Contents

Chapter 1

Introduction

Covering arrays are combinatorial objects that have been successfully applied in the design of test suites for testing systems such as software [7, 8], circuits, networks [14] and drug screening [32] where failures can be caused by the interaction between their parameters. Checking every possible interaction between all the parameters for errors is infeasible in many cases due to time and cost constraints. Thus, there is a need to generate test suites that are substantially smaller than exhaustive test suite, but highly effective in detecting faults. Pairwise testing (also called all-pairs testing or 2-way testing) is known for its effectiveness in different types of software systems [7, 17, 14]. Pair-wise testing requires that for a given numbers of input parameters to the system, each possible combination of values for any pair of parameters is covered by at least one test case. A system of 126 binary inputs with Pairwise testing would require 10 test cases; exhaustive testing would require 2^{126} test cases [22]! Pairwise testing is based on the observation that most faults are caused by interactions of at most two parameters [26]. Pairwise-generated test suites cover all combinations of two parameters, therefore, are much smaller than exhaustive ones yet still very effective in finding defects. Pairwise-generated test suites ensure that software system cannot fail due to an interaction of two parameters; however, software failures may be caused by interactions of more than two parameters. A recent NIST study indicates that failures can be triggered by interactions up to 6 parameters [16]. Here we consider the problem of generating test cases for pairwise testing (2-way interaction testing) and 3-way interaction testing. 3-way interaction testing requires that for a given number of input parameters to the system, each possible combination of values of any three parameters is covered by at least one test case. Covering arrays prove useful in locating large percentage of errors in software systems[7, 36]. The test cases are columns of a covering array. Constructions that can yield small test suites are of both theoretical and practical interest [7, 8, 17, 21, 1].

1.1 Applications of interaction testing

Testing is an important but expensive part of the software and hardware development process. It is known that more than 50% of the cost of developing a software goes to software testing. NIST [4] studies show that US economy suffers a huge loss of $59.5 billion annually due to software bugs. Through out the history, we can find many examples of faulty testing that led to disasters ranging from economic loss to even loss of human lives. Some examples of incompetent testing that led to hazardous effects are the following. In 1982, during the cold war, the CIA implemented a bug [34] in the pressure control software that the Soviet Union purchased from Canada such that it would pass the software testing that Soviet Union had then, which resulted in the explosion of the Trans-Siberian gas pipeline. An example of a hardware bug is the popular Pentium floating point division bug (1993) [11] causing errors in the division of long floating point numbers. This happened due to a flaw in the look up table employed in the division circuits and led to a loss of $475 for the Intel company. Examples of faulty interaction testing in particular are many. In 1985, the radiation therapy machine Therac-25 [18] killed three and caused severe injury to other three patients by emitting lethal doses of radiation. This was the result of a software error called race condition, where unintended events occur leading to multiple potential inputs racing to affect the output. Another example of faulty interaction testing is Europe's Ariane-5 [19] satellite (1996) that exploded seconds after its maiden flight. A register overflow happened in the processor computing the velocity passing the control over to a backup processor, which crashed as it used the same algorithm. The loss was around $500 million. The following are the main areas where pairwise testing and 3-way interaction testing find applications.

1. **Component based systems:** Component based systems [2] allow components to be developed separately which are later put together to interact to form a functional system. Components are designed with a wider range of functionality so that it can either be used independently or in composition with other components. This allows multiple implementations of components and supports

reuse and updating the already developed components, resulting in an improved threshold to changes. In the age of distributed applications, component based systems play a key role as assembling can be done without specialised skills in developing the components. Since initial cost of development and testing of components are high, this helps in reducing the overall cost as components once developed can be used as the building blocks.

As independent components are assembled to form the functional system, testing the independent components is not enough [35]. Errors might occur due to faulty interaction of various components. And therefore, there is a need to test the interactions of different components.

2. **Software testing:** When a software is designed, it is expected to meet some requirements. An effective method to validate a software at a feasible time subjected to budget constraints is one of the main concerns in software testing [25]. Since an exhaustive test suite is almost impractical, we resort to methods which optimize the cost by maximizing the code coverage. Interaction testing is found to be useful [7] in generating feasible test suites and is found to lower the cost involved substantially.

3. **Hardware testing:** The main application of interaction testing is in generating effective test suites for testing logic circuits and networks [30]. With the advancement in the technologies for improved VLSI systems in which thousands of components interact, interaction testing became an unavoidable tool in the realm of hardware testing. They also find applications in discrete device testing [3], studying interaction of factors in mobile ad hoc networks [33], etc.

4. **Design of experiments:** Consider experiments [28] which studies the variation of output with multiple input combinations. In cases where a researcher want to test that only a subset of the inputs affect the output, interaction testing can suggest a substantially smaller set of test cases.

5. **Multiple drug therapy and drug screening:** In most cases multiple drugs are simultaneously used in the treatments [15]. In such cases, the interactions between drugs occur and the cumulative effect of the drugs have to be studied before administering them. Another application of interaction testing is in drug screening. Many addictive drugs are used in medical treatment but at the same

time are abused by athletes [32]. Interaction testing helps in establishing rapid and effective methods in drug screening.

1.2 Combinatorial designs and interaction testing

From the early 1930's onwards statisticians have been using orthogonal arrays in design of experiments [12]. During the mid 1980's, Tang, Chen, and Woo [30, 31] showed that similar combinatorial objects can be appropriately applied in generating exhaustive test patterns for logic testing in circuits. In 1997, Cohen, Dalal, Freedman and Patton [7] suggested that such combinatorial designs can be applied in constructing test suites that tests all the pairwise or t-wise interactions of selected parameters at a substantially lower cost and time. This paved way to a vast area of research on interaction testing.

1.2.1 Orthogonal arrays

Orthogonal arrays were introduced by Rao [15] in 1946. They can be applied in designing experiments that estimate the main effect of parameters and their interaction effect.

Definition 1.2.1. Let g, t, k, $0 < t \leq k$ be positive integers. An *orthogonal array* on a set of g symbols, strength t, k factors and index λ denoted by $OA_\lambda(t, k, g)$ is a $k \times \lambda g^t$ array over the symbols $\{0, ..., g-1\}$ such that every $t \times \lambda g^t$ array has all g^t possible $t-$tuples occurring exactly λ times.

Often the parameter λ is ignored if $\lambda = 1$. Orthogonal arrays are computationally hard to find, but extremely useful. They have applications [15] in designing experiments, in developing error correcting codes, in determining the shelf life of drugs, in investigating the interaction of multiple drugs administered simultaneously, survey sampling, manufacturing automobiles etc. In an experiment, the rows of an orthogonal array represent the different parameters that are possible and the symbols are the multiple values that a parameter may assume. Note that the columns of an orthogonal array are the test cases.

Example 1.2.1. The following is an example of an $OA(2, 3, 2)$.

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Here, we can see that any two rows of the array cover all the four possible $2-$tuples $00, 01, 10$ and 11 exactly once.

Given parameters t, k and v, an orthogonal array need not exist. However, there might exist a two dimensional array with k rows such that any choice of t rows contain all possible $t-$tuples at least λ times. Also, Cohen et al. [7] pointed out that in the realm of software testing, it suffices to have an array that covers each t-way interaction at least once. These arrays are called covering arrays of strength t. The following section gives formal definition of a covering array.

1.3 Covering Arrays

Definition 1.3.1. A covering array $t - CA(n, k, g)$, of size n, strength t, degree k, and order g, is a $k \times n$ array on a set of g symbols with the property that in each $t \times n$ subarray, every $t \times 1$ column appears at least once.

For example, the following array is a covering array for $t = 2$, $g = 2$ and $k = 4$, because if we look at this array, we notice that whichever two rows out of the four rows are chosen, all possible pairs $00, 01, 10$ and 11 come up at least once:

$$2 - CA(5, 4, 2) = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

The columns of a covering array with $t = 2$ provides a test suite for pair-wise testing. These require a very small number of test cases compared to the total number of possible test cases. For most applications t, k and g are given and it is desirable to minimise n. The smallest possible size of a covering array for fixed parameters t, k, and g will be denoted as

$$t - CAN(k, g) = \min_{m \in \mathbb{N}} \{m \ : \ \text{there exists a } t - CA(m, k, g)\}.$$

For a good up to date survey on covering arrays, see [9]. The following example shows how the test cases are generated to test a software using a covering array. Consider the testing of an internet site that must work correctly on two operating systems, two browsers, two printers and two file formats as given below:

1. Operating systems: Windows/ Linux
2. Browsers: Internet Explorer(IE)/ Firefox
3. Printers: HP/ Epson
4. File formats: pdf/ DjVu

An exhaustive test suite consists of $2 \times 2 \times 2 \times 2\times = 16$ test configurations. However, only five test cases are enough to cover all the pair-wise interactions between different parameters. These five test cases are obtained using a $2 - CA(5, 4, 2)$.

OS	Windows	Windows	Linux	Linux	Linux
Browser	IE	firefox	IE	firefox	firefox
Printer	HP	Epson	Epson	HP	Epson
File format	pdf	DjVu	DjVu	DjVu	pdf

There is a vast array of literature [13, 6] on covering arrays, and the problem of determining small covering arrays has been studied under many guises over the past thirty years. In [13], Hartman and Raskin discussed several generalizations of the problem of creating small covering arrays motivated by their applications in the realm of software testing. One natural generalization is whether different parameters can take different number of values. There are covering arrays in which different rows can accommodate different number of symbols. These are called *mixed covering arrays.* Now we consider one of the five natural generalizations of covering arrays listed in [13].

A practical limitation in the realm of testing is budget. Given a fixed number of test cases, we consider the problem of building a testing array A with maximum possible coverage. The total number of $t-$tuples that needs to be covered for $t-$way interactions is $\binom{k}{t}g^t$. The coverage measure of a testing array A of strength t is defined by

$$\mu_t(A) = \frac{N_t(A)}{\binom{k}{t}g^t}$$

where $N_t(A)$ is the number of distinct $t-$tuples covered in the columns of A. Our objective is to construct a testing array A of size at most n having largest possible coverage measure, given fixed values of t, k, g and n. This problem is called *covering arrays with budget constrains.*

1.3.1 Construction of covering arrays

Covering arrays are generally found through greedy algorithms, heuristic searches and algebraic constructions assisted by algorithmic techniques. Exhaustive searches can always give the smallest test suite but it is infeasible. In such cases, efficient heuristic techniques help in obtaining results that are close to the optimum. The most popular greedy algorithm is the Automatic Efficient Test Generator (AETG) which was developed by Cohen et al [7].

There are three main algebraic methods that are used. The earliest is the method of finite field construction [15] for orthogonal arrays which are covering arrays too. Block recursive construction [29] develops a covering array $2-CA(n+m, rs, g)$ from two covering arrays $2-CA(n, r, g)$ and $2-CA(m, s, g)$. The most recent technique is the group construction method [24] which is an algebraic method assisted by computer search.

1.4 Overview of the thesis

The aim of the thesis is to propose techniques for constructing good covering arrays using group theory coupled with exhaustive and heuristic computer search. A key advantage of these methods is that we search for a small vector that can be used to construct a covering array, rather than searching for an entire array. Chapter 2 describes the group construction method developed by Meagher and Stevens [24] for covering arrays of strength two and illustrates the method using an example. This method employs the action on the symbols of a group of order $g-1$ fixing one symbol. In Chapter 3, we extend this method so that the number of fixed symbols f is permitted to take any non-negative integer value and group of order $g-f$ on symbols is simply $\mathbb{Z}_{g-f}$, for constructing covering arrays of strength two. This method produces covering arrays which are smaller in size compared to those which are found by heuristic tools like NIST IPOG-F etc. In Chapter 4, we generalize this technique for constructing covering arrays of strength 3 with budget constraints.

Chapter 2

Group construction of covering arrays

2.1 Introduction

In this chapter, we describe the group construction method developed by Meagher and Stevens [24] for covering arrays of strength two. It is based on the algebraic method developed by Chateauneuf, Colbourn and Kreher [6] for covering arrays of strength three using graph factorization. For many values of k between $g+1$ and $2g+1$, group construction produces covering arrays of size $n = k(g-1)+1$ which is better than some of the best known upper bounds for covering arrays.

2.2 The Group construction

Group construction requires selecting an appropriate group $G < Sym_g$ and a vector $v \in \mathbb{Z}_g^k$ called a *starter vector*. The choice of a starter vector depends on the group G and is selected through an algorithmic search. A circulant matrix M is created using the vector v. If $x \in G$, then M^x is the $k \times k$ matrix where $[i,j]$ entry is $M[i,j]^x$, the image of $M[i,j]$ under x. The matrix obtained by developing M by G is the $k \times k|G|$ matrix

$$M^G = [M^x : x \in G].$$

A small array C is often added to complete the covering conditions.

2.3 Selecting a starter vector

For a vector v to be a starter vector, any two rows in the matrix M must have at least one element from each of the orbits of the group action of G on the pairs from $\mathbb{Z}_g$. Starter vectors are found by exhaustive searches which look for the following properties. Define for each $0 < i < k$, the set

$$d_i = \{(v_j, v_{j+i}) | j = 0, 1, ..., k-1\},$$

where the subscripts are taken modulo k. A vector $v \in \mathbb{Z}_g^k$ is a starter vector if each set d_i has representation from each orbit of the group action of G on the pairs from $\mathbb{Z}_g$.
The following example illustrates the group construction method.

Example 2.3.1. In this example, we construct a $2 - CA(11, 5, 3)$ by the group construction method. Let $G = \{e, (12)\}$ and consider the vector $v = (01112) \in Z_3^5$. The sets d_i are as follows:

1. $d_1 = \{(0,1), (1,1), (1,2), (2,0)\}$
2. $d_2 = \{(0,1), (1,1), (1,2), (1,0), (2,1)\}$
3. $d_3 = \{(0,1), (1,2), (1,0), (1,1), (2,1)\}$
4. $d_4 = \{(0,2), (1,0), (1,1), (1,1), (2,1)\}$

The orbits of the group action of $G = \{e, (12)\}$ on the pairs from $\mathbb{Z}_3$ are :

1. $O_1 = (0,0)$
2. $O_2 = (0,1), (0,2)$
3. $O_3 = (1,0), (2,0)$
4. $O_4 = (1,1), (2,2)$
5. $O_5 = (1,2), (2,1)$

Note that G acts on $1, 2 \in \mathbb{Z}_3$ and fixes $0 \in \mathbb{Z}_3$. We can verify that each d_i has representation from every orbit except the orbit 1 and hence v is a starter vector. Once a starter vector v is found, produce the circulant matrix M from $v = (01112)$.

$$M = \begin{pmatrix} 0 & 2 & 1 & 1 & 1 \\ 1 & 0 & 2 & 1 & 1 \\ 1 & 1 & 0 & 2 & 1 \\ 1 & 1 & 1 & 0 & 2 \\ 2 & 1 & 1 & 1 & 0 \end{pmatrix}$$

The action of G on M produces M^e and $M^{(12)}$ as follows:

$$M^e = \begin{pmatrix} 0 & 2 & 1 & 1 & 1 \\ 1 & 0 & 2 & 1 & 1 \\ 1 & 1 & 0 & 2 & 1 \\ 1 & 1 & 1 & 0 & 2 \\ 2 & 1 & 1 & 1 & 0 \end{pmatrix} \qquad M^{(12)} = \begin{pmatrix} 0 & 1 & 2 & 2 & 2 \\ 2 & 0 & 1 & 2 & 2 \\ 2 & 2 & 0 & 1 & 2 \\ 2 & 2 & 2 & 0 & 1 \\ 1 & 2 & 2 & 2 & 0 \end{pmatrix}.$$

The matrices $C = (0, ..., 0)^T$, M^e and $M^{(12)}$ are concatenated to build a covering array $2 - CA(11, 5, 3)$.

$$\left(\begin{array}{ccccc|ccccc|c} 0 & 2 & 1 & 1 & 1 & 0 & 1 & 2 & 2 & 2 & 0 \\ 1 & 0 & 2 & 1 & 1 & 2 & 0 & 1 & 2 & 2 & 0 \\ 1 & 1 & 0 & 2 & 1 & 2 & 2 & 0 & 1 & 2 & 0 \\ 1 & 1 & 1 & 0 & 2 & 2 & 2 & 2 & 0 & 1 & 0 \\ 2 & 1 & 1 & 1 & 0 & 1 & 2 & 2 & 2 & 0 & 0 \end{array}\right)$$

The group construction method coupled with computer search substantially improved the upper bounds for strength two covering arrays in many cases. In cases where a starter vector was not found, the method could suggest a good candidate for a seed in heuristic search. Extension of the group construction method has also been used to find strength three covering arrays with budget constraints [23]. The advantage of group construction method over any previous algebraic construction is that it suffices to look for a starter vector than for an entire array or a block of the array. This made it a tool easy to be implemented in shorter time.

Over the years researchers have come up with variants of the group construction method that improved the best known covering array numbers. The following two chapters discuss the method of fixing symbols in group construction for covering arrays of strength two and three. It is derived from the method given by J. R Lobb et. al. for covering arrays of strength two. Group construction method employs the action on the symbols of a group of order $g-1$ fixing one symbol. In Chapter 3, we extend this method so that the number of fixed symbols f is permitted to take any non-negative integer value.

Chapter 3

Covering arrays of strength two

3.1 Introduction

In this chapter, we propose a construction method for covering arrays of strength two that combines an algebraic construction with a computer search. See also [5]. The construction given here follows the group construction method used by Meagher and Stevens in [24] and Lobb et. al. in [20]. A key advantage of group construction method is that it searches for a small vector that can be used to construct a covering array rather than for an entire array.

The group construction method employs the group action of the symbols of a group of order $g - 1$ fixing one symbol. Here we extend this method so that the number of fixed symbols can take any non-negative integer value f. In the group construction method, to construct a covering array on g symbols, we use a group of size $g - 1$ and attach a constant column for the fixed symbol. When the number of fixed symbols can take any non-negative integer value f, it suffices to use a group of order $g - f$, thereby requiring only $g - f$ matrices to be concatenated. However, we have to add a small matrix of size n' to complete the covering conditions. If $n' \leq f \times k$, our method may lead to an improved covering array number. The proposed method is found to produce covering arrays which are smaller in size compared to those which are found by heuristic tools like NIST IPOG-F etc.

3.2 The Fixed Symbols Construction

Let X be the set of g symbols on which we are to construct a $2-CA(n,k,g)$. Let G be a group acting on the set X. The group G acts on $|G|$ points and fixes the remaining $f = g - |G|$ points. Let $F = \{\infty_0, \infty_1, ..., \infty_{f-1}\}$ be the set of fixed symbols. Then $X = \{g_1, g_2, \ldots, g_{|G|}\} \cup \{\infty_0, \infty_1, ..., \infty_{f-1}\}$. The number of fixed symbols can be any nonnegative integer value. The action of G on pairs from X has six orbits. These six orbits are determined by the pattern of the entries in their 2-tuples:

1. $\{(\infty_i, \infty_i)^T \ : \ \infty_i \in F\}$
2. $\{(\infty_i, \infty_j)^T \ : \ \infty_i, \infty_j \in F, \infty_i \neq \infty_j\}$
3. $\{(\infty_i, g_j)^T \ : \ \infty_i \in F, g_j \in G\}$
4. $\{(g_j, \infty_i)^T \ : \ \infty_i \in F, g_j \in G\}$
5. $\{(g_i, g_j)^T \ : \ g_i, g_j \in G, g_i \neq g_j\}$
6. $\{(g_i, g_i)^T \ : \ g_i \in G\}$

Our construction requires selecting a vector $v \in X^k$, called a starter vector. We use the vector v to form a $k \times k$ circulant matrix M. For v to be a starter vector, any two rows in the matrix M must have at least one element from each of the orbits $3-6$. We will also need an array C to complete the covering conditions, where C is $2-CA(n', k, f)$ with symbols from F. The group acting on the matrix M produces several matrices that are concatenated with C to form a covering array. We give an example to illustrate the method.

Example 3.2.1. Let $k = 9$ and $g = 4$. Let $G = \mathbb{Z}_2$ and $v = \infty_0\infty_0 0\infty_1 011\infty_1 0$.

Here $F = \{\infty_0, \infty_1\}$. Create the following circulant matrix from v,

$$M = \begin{pmatrix} \infty_0 & 0 & \infty_1 & 1 & 1 & 0 & \infty_1 & 0 & \infty_0 \\ \infty_0 & \infty_0 & 0 & \infty_1 & 1 & 1 & 0 & \infty_1 & 0 \\ 0 & \infty_0 & \infty_0 & 0 & \infty_1 & 1 & 1 & 0 & \infty_1 \\ \infty_1 & 0 & \infty_0 & \infty_0 & 0 & \infty_1 & 1 & 1 & 0 \\ 0 & \infty_1 & 0 & \infty_0 & \infty_0 & 0 & \infty_1 & 1 & 1 \\ 1 & 0 & \infty_1 & 0 & \infty_0 & \infty_0 & 0 & \infty_1 & 1 \\ 1 & 1 & 0 & \infty_1 & 0 & \infty_0 & \infty_0 & 0 & \infty_1 \\ \infty_1 & 1 & 1 & 0 & \infty_1 & 0 & \infty_0 & \infty_0 & 0 \\ 0 & \infty_1 & 1 & 1 & 0 & \infty_1 & 0 & \infty_0 & \infty_0 \end{pmatrix}$$

The elements of $\mathbb{Z}_2 = \{0, 1\}$ acting on M produce $M^0 = M$ and

$$M^1 = \begin{pmatrix} \infty_0 & 1 & \infty_1 & 0 & 0 & 1 & \infty_1 & 1 & \infty_0 \\ \infty_0 & \infty_0 & 1 & \infty_1 & 0 & 0 & 1 & \infty_1 & 1 \\ 1 & \infty_0 & \infty_0 & 1 & \infty_1 & 0 & 0 & 1 & \infty_1 \\ \infty_1 & 1 & \infty_0 & \infty_0 & 1 & \infty_1 & 0 & 0 & 1 \\ 1 & \infty_1 & 1 & \infty_0 & \infty_0 & 1 & \infty_1 & 0 & 0 \\ 0 & 1 & \infty_1 & 1 & \infty_0 & \infty_0 & 1 & \infty_1 & 0 \\ 0 & 0 & 1 & \infty_1 & 1 & \infty_0 & \infty_0 & 1 & \infty_1 \\ \infty_1 & 0 & 0 & 1 & \infty_1 & 1 & \infty_0 & \infty_0 & 1 \\ 1 & \infty_1 & 0 & 0 & 1 & \infty_1 & 1 & \infty_0 & \infty_0 \end{pmatrix}$$

We also need to use $C = 2 - CA(6, 9, 2)$, a covering array with entries from $F = \{\infty_0, \infty_1\}$ to ensure the coverage of all pairs of the types 1-2. By horizontally concatenating the matrices C, M^0, and M^1, we get a $2 - CA(24, 9, 4)$.

3.2.1 Selecting a Starter Vector v

For v to be a starter vector, any two rows in the matrix M must have at least one element from each of the orbits $3 - 6$. The idea is first to decide the positions of the fixed symbols ∞_i's in v such that the array covers all the interactions of the types 3 and 4 and then fill in the remaining positions with the symbols from G so that the array covers orbits of the types 5 and 6.

Proposition 3.2.1. *Let the rows of an array be indexed by the additive group $R = \mathbb{Z}_k$.*

Let $G = \mathbb{Z}_{g-f}$ and $F = \{\infty_0, \infty_1, ..., \infty_{f-1}\}$. If there exists a partition $R_0, R_1, \dots, R_{f-1}, \bar{R}$ of R such that

$$R \backslash \{0\} = \{b - a | a \in R_i, b \in \bar{R}\} \text{ for all } R_i$$

then there exists a $k \times k|G|$ array on g symbols $G \cup F$so that for every two distinct rows r_1 and r_2, and every two elements $\gamma \in G$, $\infty \in F$, there exists a column c in which the entry in cell $[r_1, c]$ is ∞ and that in cell $[r_2, c]$ is γ.

Proof: Suppose such a partition of R exists. Construct a vector v as follows: if $a \in R_i$, $0 \leq i < f$, then place ∞_i in the ath position and otherwise if $a \in \bar{R}$, place an element of G in the ath position. We use the vector to form a $k \times k$ circulant matrix M. If $x \in \mathbb{Z}_{g-f}$, then M^x is the $k \times k$ matrix whose $[i, j]$ entry is $M[i, j]^x$, the group action of x on $M[i, j]$. The matrix obtained by concatenating the $g - f$ matrices formed by the group action of $G = \mathbb{Z}_{g-f}$ on M is the $k \times k|G|$ matrix

$$M^G = [M^x \ : \ x \in G].$$

Given any two row indices $r_1, r_2 \in R$, there exist $a \in R_i$ and $b \in \bar{R}$ such that $r_2 - r_1 = b - a \bmod k$. By the construction of vector v, the element in the ath position of vector v is ∞_i and that in the bth position is some $\gamma' \in G$. For $\gamma' \in G$, there exists an element $x \in \mathbb{Z}_{g-f}$ such that $\gamma = \gamma' + x \pmod{g - f}$. Consider the matrix M^x produced by the group action of x on M. The first column of M^x has ∞_i in row a and γ in row b. Now, by the construction of the circulant matrix, we can always find a column c in the array M^x such that $M^x(r_1, c) = \infty_i$ and $M^x(r_2, c) = \gamma$. Hence the proposition follows.

□

Once the positions of the fixed symbols are determined in vector v, any interaction of the form (γ, ∞) and (∞, γ) are covered in C irrespective of which non-fixed element occupy the position a for $a \in \bar{R}$. Now we fill the remaining positions of v with entries from G such that, any two rows in the matrix M must have at least one element from the orbits $5 - 6$. Consider the sets D_i

$$D_i = \{(v_j, v_{j+i}) \mid j = 0, 1, \dots, k - 1\},$$

subscripts taken modulo k. For v to be a starter vector, fill the remaining positions of v using entries from G such that each set D_i, for $i = 1, \dots, k - 1$, must contain a representative from orbits $5-6$. Some sample programs are given in the Appendix Section A.

3.3 Results

Stater vectors v are found by computer search. Starter vectors exists for many values of k and g. In the cases where the number of fixed symbols f could have more than one value satisfying Proposition 3.2.1, we simply choose the one that gives us smaller size covering array. Table 1 lists starter vectors for several values of k and g. A comparison of our fixed symbols construction with heuristic tools like NIST IPOG-F [10] in Table 1 shows that the fixed symbols construction produces significantly smaller size covering arrays.

Table 3.1: A comparison of the number of test cases n generated by the Fixed Symbols Construction and NIST IPOG-F [10].

g	k	f	Starter vector	Fixed Symbols Construction: n	NIST IPOG-F [10]
3	5	1	$0\infty_0 001$	11	13
3	6	1	$0\infty_0 0001$	13	15
3	7	1	$0\infty_0 00001$	15	15
4	5	1	$0\infty_0 011$	16	22
4	6	1	$0\infty_0 0011$	19	24
4	7	1	$0\infty_0 00011$	22	27
4	9	2	$\infty_0\infty_0 0\infty_1 011\infty_1 0$	24	29
4	10	2	$0001\infty_0 0\infty_1 1\infty_0\infty_1$	26	29
4	11	2	$0\infty_1 0\infty_1 000\infty_0\infty_0 10$	29	31
5	10	2	$0\infty_0 100\infty_1\infty_1 20\infty_0$	36	45
5	11	2	$0\infty_1 0\infty_1 011\infty_0\infty_0 11$	40	46
5	12	2	$000\infty_0 0\infty_0\infty_1 1012\infty_1$	43	48
5	13	2	$\infty_1 00\infty_0\infty_0 000101\infty_1 1$	46	49
5	14	2	$0000001\infty_0 0\infty_1\infty_1 1\infty_0 1$	49	50
6	9	1	$0\infty_0 0010231$	46	52
6	10	1	$0\infty_0 00103013$	51	63
6	11	1	$0\infty_0 000010231$	56	64
6	12	1	$0\infty_0 0000010231$	61	67
6	13	1	$0\infty_0 00000010231$	66	68
7	10	1	$0\infty_0 00231023$	61	NA
7	11	1	$0\infty_0 000103522$	67	NA
7	12	1	$0\infty_0 0000154304$	73	NA
7	13	2	$\infty_1 01\infty_0\infty_0 224210\infty_1 4$	72	NA
7	14	2	$001034154\infty_1\infty_1 3\infty_0 4$	77	NA
7	15	2	$0000\infty_1 143\infty_1 23\infty_0 0\infty_0 3$	82	NA
7	17	3	$00\infty_2 1\infty_0 21\infty_1 2\infty_2\infty_0\infty_1\infty_2 0120$	83	NA
8	11	1	$0\infty_0 001422410$	78	NA
8	15	2	$0010\infty_1 241\infty_1 13\infty_0 0\infty_0 2$	97	NA
8	16	2	$\infty_1\infty_1 0000253\infty_0 0545\infty_0 2$	104	NA
8	18	3	$00000134\infty_1\infty_2\infty_0\infty_1 204\infty_0 3\infty_2$	104	NA
8	19	3	$00\infty_0 0000\infty_2 1\infty_2\infty_1 24042\infty_1\infty_0 1$	107	NA

Chapter 4

Covering arrays of strength three

4.1 Introduction

In this chapter, we propose an algebraic technique for constructing covering arrays of strength three with budget constraints.

In most software development environments, the time, computing and human resources needed to perform the testing of a component is strictly limited [13]. Thus an interesting and practical problem is that of finding a testing array or testing suite with maximum coverage, given a fixed budget for executing a maximum of n test cases (number of columns of the testing array).

The coverage measure $\mu_3(A)$ of a testing array A is defined as the ratio between the number of distinct $3-$tuples contained in the column vector of A and the total number of $3-$tuples. That is,

$$\mu_3(A) = \frac{N_3(A)}{\binom{k}{3}g^3},$$

where $N_3(A)$ is the number of distinct $3-$tuples contained in the column vector of A.

The *covering arrays with budget constraints* problem is to construct a testing array A of size at most n having largest possible coverage measure, given fixed values of t, k and g. Here we consider $t = 3$. The construction given here is a generalization of the algebraic method given by Meagher and Stevens [24].

In the following section, we summarize the results from group theory that we use.

4.1.1 Simple Linear Group

$GF(s)$ is the Galois field where $s = p^m$ for some prime p. Let $H(s)$ be the set of all functions of the form $x \mapsto ax + b, a \neq 0$, where $a, b \in GF(s)$. This set $H(s)$ forms a group under the operation of functional transformation called the *simple linear group* [27] and the action of $H(s)$ on the elements of $GF(s)$ is $2-$transitive. The order of $H(s)$ is $s(s-1)$.

4.2 The Fixed Symbol Construction

Given k, g and n, we present a method to construct a $k \times n$ testing array A on g symbols with maximum possible coverage measure. Choose g such that $g-1$ is a prime or prime power. Let $X = GF(g-1) \cup \{\infty\}$ be the set of g symbols on which we are to construct a testing array. We select a group G such that the action of G on $X = GF(g-1) \cup \{\infty\}$ is sharply $2-$transitive. Here we choose G to be $H(g-1)$. Under this group action, there are precisely $g+11$ orbits of $3-$tuples. They are listed below:

1. $\{(\infty, \infty, \infty)^T\}$
2. $\{(\infty, \infty, x)^T : x \in GF(g-1)\}$
3. $\{(\infty, x, \infty)^T : x \in GF(g-1)\}$
4. $\{(x, \infty, \infty)^T : x \in GF(g-1)\}$
5. $\{(\infty, x, x)^T : x \in GF(g-1)\}$
6. $\{(x, \infty, x)^T : x \in GF(g-1)\}$
7. $\{(x, x, \infty)^T : x \in GF(g-1)\}$
8. $\{(\infty, x, y)^T : x, y \in GF(g-1), x \neq y\}$
9. $\{(x, \infty, y)^T : x, y \in GF(g-1), x \neq y\}$
10. $\{(x, y, \infty)^T : x, y \in GF(g-1), x \neq y\}$
11. $\{(x, x, x)^T : x \in GF(g-1)\}$

12. $\{(x,x,y)^T : x,y \in GF(g-1), x \neq y\}$

13. $\{(x,y,x)^T : x,y \in GF(g-1), x \neq y\}$

14. $\{(y,x,x)^T : x,y \in GF(g-1), x \neq y\}$

15. $g-3$ orbits of patterns with 3 distinct entries from $GF(g-1)$.
    $\{(x,y,z)^T : x,y,z \in G, x \neq y \neq z\}$

Our construction involves selecting a vector $v \in X^k$. We use the vector v to form a $k \times k$ circulant matrix M. If $h \in H(g-1)$, then M^h is the $k \times k$ matrix where the $[i,j]$ entry is $M[i,j]^h$, the image of $M[i,j]$ under h. The matrix obtained by developing M by $H(g-1)$ is the $k \times k(g-1)(g-2)$ matrix

$$M^{H(g-1)} = [M^h : h \in H(g-1)]$$

Let C=$(\infty, \infty, ..., \infty)^T$. The goal is to choose a vector v so that the matrix $[M^{H(g-1)}, C]$ is a covering array or a testing array with high coverage measure. A vector $v \in X^k$ is said to be a *starter vector* if any $3 \times k$ subarray of the circulant matrix M has at least one representative from each of the orbits $2-15$.

If a starter vector v is found, $[M^{H(g-1)}, C]$ is a covering array $3-CA(k(g-1)(g-2), k, g)$. If a starter vector is not found, we look for a vector that produces an array $[M^{H(g-1)}, C]$ with maximum possible coverage measure.

Example 4.2.1. Let $g = 3$, $k = 21$, and $n = 45$. Then $X = GF(2) \cup \{\infty\}$ and $G = H(2)$. Let $v = (0000\infty0\infty1101\infty\infty\infty\infty\infty11\infty010100\infty111\infty\infty\infty001)$. Build the following circulant matrix M from v:

$$M = \begin{pmatrix}
\infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 \\
0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty \\
\infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 \\
1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 \\
1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty \\
\infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty \\
\infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 \\
1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 \\
1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty \\
\infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty \\
\infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 \\
0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty \\
\infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 \\
0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 & \infty \\
\infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 & 1 \\
1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 & 0 \\
0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 & 1 \\
1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 & 0 \\
0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty & 0 \\
0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty & \infty \\
\infty & 0 & 0 & 1 & 0 & 1 & \infty & 0 & \infty & 0 & \infty & \infty & 1 & 1 & \infty & \infty & 1 & 1 & \infty & 0 & \infty
\end{pmatrix}$$

The action of the group $H(2) = \{x, x+1\}$ produces the two matrices $M^x = M$ and M^{x+1} which are concatenated to give the matrix $M^{H(2)}$. A small array C as shown also needs to be attached to cover interaction in orbit 11 and orbit 1.

$$C = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty
\end{pmatrix}^T$$

The matrix $[M^{H(2)}, C]$ is a 21×45 testing array with coverage measure 0.908772.

4.2.1 Selecting a starter vector

A vector $v \in X^k$ is a starter vector if any three rows of the corresponding circulant matrix M contains at least one representative from each of the orbits $2 - 15$. M is called starter matrix. First we fix the positions of the fixed symbol ∞ in v such that any three rows of the matrix M cover at least one representative from each of the orbits $2-10$ and then fill the remaining positions of v with symbols from $GF(g-1)$ so that any three rows of M contain at least one representative from the orbits $11-15$. The following two propositions provide us criteria to decide the positions of the fixed symbol ∞ in the vector v.

Proposition 4.2.1. *Let the rows of a $k \times k$ starter matrix M be indexed by the elements of the additive group $R = \mathbb{Z}_k$. If $\exists$ a partition $R_0, \bar{R}$ of R such that*

$$R \times R \setminus \{(r,r),(0,r),(r,0) : r \in R\} = \{(b-a, c-a) | a, b \in R_0, c \in \bar{R}\},$$

then there exists a $k \times k|H(g-1)|$ array A on g symbols X so that for any three distinct rows r_1, r_2, r_3 and every entry (∞, ∞, x), there exists a column l such that $A[r_1, l] = \infty$,$A[r_2, l] = \infty$ and $A[r_3, l] = x$.

Proof: Suppose such a partition of R exists. Construct a vector v such that when $i \in R_0$, insert ∞ in the ith position and whenever $i \in \bar{R}$ insert an element of $y \in GF(g-1)$ in the ith position. Produce the $k \times k$ circulant matrix (starter matrix) M using the vector v. If $x \in H(g-1)$, then M^x is the $k \times k$ matrix whose $[i,j]$ entry is $M[i,j]^x$, the group action of x on $M[i,j]$. The matrix obtained by concatenating the $(g-1)(g-2)$ matrices formed by the group action of $H(g-1)$ on M is the $k \times k(g-1)(g-2)$ matrix $M^{H(g-1)} = [M^x : x \in H(g-1)]$. Let $A = M^{H(g-1)}$.

Given any three row indices r_1, r_2, r_3, there exists $a, b \in R_0$ and $c \in \bar{R}$ such that $(b-a, c-a) = (r_2 - r_1, r_3 - r_1)$ where all arithmetics are taken modulo k. In vector v, ∞ is placed in the ath and bth positions and an element $y \in GF(g-1)$ is placed in the cth position. For $x, y \in GF(g-1)$ there exists an element $z \in H(g-1)$ such that $y \cdot z = x$. Look at the matrix M^z obtained by the action of $z \in H(g-1)$ on the matrix M. The first column of M^z has ∞ in rows a and b and x in row c. As M^z is a circulant matrix, we can find a column l in M^z such that $M^z[r_1, l] = \infty$, $M^z[r_2, l] = \infty$ and $M^z[r_3, l] = x$. □

If a partition of R satisfying Proposition 1 is found, then any three rows of the matrix A obtained from associated vector v cover all $3-$tuples from orbits $2-4$.

Proposition 4.2.2. *Let the rows of a $k \times k$ starter matrix M be indexed by the elements of the additive group $R = \mathbb{Z}_k$. If $\exists$ a partition $R_0, \bar{R}$ of R such that*

$$R \times R \setminus \{(r,r),(0,r),(r,0) : r \in R\} = \{(b-a, c-a) | a \in R_0, b, c \in \bar{R}\},$$

then there exists a $k \times k|H(g-1)|$ array A on g symbols X so that for any three distinct rows r_1, r_2, r_3 and every $3-$tuple of the form (∞, x, y) or (∞, x, x), there exists a column l such that $A[r_1, l] = \infty$, $A[r_2, l] = x$, $A[r_3, l] = y$ or $A[r_1, l] = \infty$, $A[r_2, l] = x$, $A[r_3, l] = x$.

Proof: The proof is similar to that of Proposition 1. Suppose such a partition of R exists. Construct a vector v such that when $i \in R_0$, insert ∞ in the ith position and whenever $i \in \bar{R}$ insert an element of $\in GF(g-1)$ in the ith position. Given any three row indices $r_1, r_2, r_3 \in R$, there exists $a \in R_0$ and $b, c \in \bar{R}$ such that $(b-a, c-a) = (r_2 - r_1, r_3 - r_1) \bmod k$. In vector v, the entry at position a is ∞ and the entries at positions b and c are $x' \in GF(g-1)$ and $y' \in GF(g-1)$ respectively. Let $A = M^{H(g-1)}$.
Case 1: Let $x' \neq y'$. As the action of $H(g-1)$ on $GF(g-1)$ is sharply 2-transitive, there exist a unique element $z \in H(g-1)$ such that $x' \cdot z = x$ and $y' \cdot z = y$. The first column of the matrix M^z has ∞ in row a and entries x and y in rows b and c respectively. As M^z is a circulant matrix, we can always find a column l in M^z such that $M^z[r_1, l] = \infty$, $M^z[r_2, l] = x$ and $M^z[r_3, l] = y$.
Case 2: Let $x' = y'$. As the action of $H(g-1)$ on $GF(g-1)$ is transitive, there exist an element $z \in H(g-1)$ such that $x' \cdot z = x$. The first column of the matrix M^z has ∞ in row a and entry x in rows b and c. As M^z is a circulant matrix, we can always find a column l in M^z such that $M^z[r_1, l] = \infty$, $M^z[r_2, l] = x$ and $M^z[r_3, l] = x$. □

If a partition of R satisfying Proposition 2 is found, then any three rows of the matrix A, obtained from vector v, covers all $3-$ tuples from orbits $5-7$ or $8-10$.
Let R_0 and $\bar{R}$ be a partition of R satisfying Proposition 1 and Proposition 2. A vector v is constructed such that when $i \in R_0$, insert ∞ in the ith position. Then any $3-$ rows of the associated starter matrix M contains at least one $3-$tuple from the orbits $2-7$ or orbits $2-4$ and $8-10$.

Once the positions of the fixed symbols are determined in the vector v, we fill the remaining positions with entries from $GF(g-1)$ such that any three rows of M contains at least one element from the orbits $12-15$. Consider the sets

$$d[x, y] = \{(v_i, v_{i+x}, v_{i+x+y}) : i = 0, 1, ..., k-1\}$$

For v to be a starter vector or a starter vector with good coverage, fill the remaining positions of v using entries from $GF(g-1)$ such that each set $d[x, y]$ contains a representative from orbits $12-15$. We now give specific choices of x and y and the number of disjoint $d[x,y]$ sets.

4.2.2 Choice of $d[x,y]$ classes

A covering array of strength 3 satisfies the property that for any three distinct rows all possible 3-tuples of g symbols occur atleast once as a column. We can divide the collection of $\binom{k}{3}$ choices of three distinct rows from k rows into classes, putting two choices (α,β,γ) and (α',β',γ') in the same class if $\beta-\alpha=\beta'-\alpha'$ mod k and $\gamma-\beta=\gamma'-\beta'$ mod k. We define the class $[x,y]$ as follows:

$$[x,y]=\{(i,i+x,i+x+y) \bmod k \mid i=0,1,\ldots,k-1\}.$$

Suppose there are ℓ classes in such a division. Since each of these classes contains exactly k choices, $k\ell=\binom{k}{3}$ and so $\ell=\frac{\binom{k}{3}}{k}$ giving

$$\ell=\frac{(k-1)(k-2)}{6}.$$

The number of classes $\ell=\frac{(k-1)(k-2)}{6}$ is an integer when k is not a multiple of 3. It is easy to see that when k is not a multiple of 3, the distinct classes are $[x,y]$ where

$$\begin{aligned} x &= 1,2\cdots\left[\frac{k-1}{3}\right] \quad \text{and} \\ y &= x,x+1\cdots k-1-2x \end{aligned}$$

The number of classes $\ell=\frac{(k-1)(k-2)}{6}$ is never an integer when $k\equiv 0 \pmod 3$. Thus, the number of classes is $\ell=\lfloor\frac{(k-1)(k-2)}{6}\rfloor+1$ when k is a multiple of 3. Here, along with the classes $[x,y]$ mentioned above, we also consider the class $[\frac{k}{3},\frac{k}{3}]$ which contains only $\frac{k}{3}$ choices.

Example 4.2.2. Let $k=7$. Then the number of classes $\ell=5$ and the classes are $[1,1]$, $[1,2]$, $[1,3]$, $[1,4]$, and $[2,2]$, and each of these classes contains 7 choices. Thus these five classes include all $5\times 7=\binom{7}{3}$ choices.

Example 4.2.3. Let $k=9$. Then the number of classes $\ell=10$ and the classes are $[1,1]$, $[1,2]$, $[1,3]$, $[1,4]$, $[1,5]$, $[1,6]$, $[2,2]$, $[2,3]$, $[2,4]$, and $[3,3]$. Note that $[3,3]=\{(0,3,6),(1,4,7),(2,5,8)\}$ contains only three choices whereas the other classes contain nine choices each. Thus these ten classes altogether include all $9 \times 9 + 3 = \binom{9}{3}$ choices.

The following algorithm generates the $[x, y]$ classes for a given k.

```
EQUIVALENCE-CLASSES($k$)
Input: $k$
Output: All $[x, y]$ classes.
for x← $1$ to $k$ do

  for y← $x$ to $k$ do

    if $x + y \leq k - 2$ and $((k - y) + (k - x))\%k > x$ or $k\%3 == 0$ and $x == \frac{k}{3}$
    and $y == \frac{k}{3}$ then
      add $[x, y]$
    end if
  end for
end for
```

For sample programs, see Appendix Section B.

4.3 Results

For computational convenience, we rewrite the coverage measure in terms of classes $[x, y]$ and $d[x, y]$ as follows:

$$\mu_3(A) = \frac{\sum\limits_{x,y} |[x, y]| \times \text{number of distinct 3-tuples covered by } d[x, y,]}{\binom{k}{3} g^3}.$$

We use computer search to find vectors v with large coverage measure. Table 4.1 shows R_0, vectors with high coverage, the number of test cases generated by our method and best known n with full coverage for $g = 3$. A comparison of our construction with best known covering arrays shows that our construction provides smaller testing arrays with good coverage measure.

Table 4.1: A comparison of the number of test cases (n) produced by our construction with high coverage measure and best known n for full coverage for strength three covering arrays on 3 symbols.

k	Vector v with good coverage	Our Results	Best
		n (coverage measure)	n [9]
16	0∞∞0∞∞1∞1110∞∞∞1	35 (0.79476)	51
17	0011∞∞101∞∞1∞∞∞1∞	37 (0.82956)	58
18	0∞0∞∞∞10∞0∞∞10∞110	39 (0.870915)	59
19	∞010∞∞0011∞1∞∞∞0∞10	41 (0.899782)	59
20	∞0∞101110∞1∞∞0∞∞∞10∞	43 (0.883041)	59
21	∞∞00101∞0∞0∞∞11∞∞11∞0	45 (0.908772)	66
22	∞0∞0∞∞10∞1∞∞∞0111∞0100	47 (0.930159)	66
23	0∞10010∞∞1∞0∞∞1∞∞111∞00	49 (0.949976)	69
24	000∞∞∞010∞1∞∞∞110011∞10∞	51 (0.951691)	71
25	∞0011∞∞1101∞10∞001∞0∞∞∞10	53 (0.973027)	72
26	0∞∞01011100∞1∞∞10∞1∞1∞01∞0	55 (0.975556)	72
27	0∞∞01011100∞1∞∞10∞1∞1∞01∞0	57 (0.975385)	75
28	0∞∞01011100∞1∞∞10∞1∞1∞01∞0	59 (0.98164)	75
29	0∞∞01011100∞1∞∞10∞1∞1∞01∞0	61 (0.983539)	81
30	0∞∞01011100∞1∞∞10∞1∞1∞01∞0	63 (0.985769)	81
31	00001∞∞0∞∞∞01∞0∞1∞∞1010∞11100∞0	65 (0.985696)	81
32	∞∞0∞0∞0∞11∞0010∞0100∞∞01∞010111∞	67 (0.989964)	81
33	∞∞001100∞0100100∞1∞∞∞1∞10∞01∞∞111	69 (0.994624)	88
34	0000∞0∞1101∞∞∞∞∞11∞010100∞111∞∞001	71 (0.994108)	88
35	∞0∞0∞000100111∞11∞∞1∞01∞1∞∞00∞∞1011	73 (0.992078)	89
36	∞∞∞00000110001∞10∞∞1010∞10∞∞1∞0∞11∞1	75 (0.996265)	89
37	000000∞110100∞0∞∞∞00∞101∞∞1∞00∞1110∞∞	77 (0.994709)	89
38	∞∞00∞100∞∞100∞0∞111∞10101∞1000∞10110∞∞	79 (0.99633)	89
39	0∞0010∞00∞1001∞11010∞∞0010111∞0∞∞11∞∞1∞	81 (0.99431)	89
40	00010111∞1∞011∞∞∞∞∞100∞0∞∞0∞0011∞0011∞11	83 (0.995502)	89
41	∞0∞00∞01∞101001∞∞∞1111000∞1100∞000∞10∞∞∞1	85 (0.998576)	93
42	∞010010∞0∞1∞∞1∞∞1∞110010∞0∞01100011∞0∞000∞	87 (0.999187)	93
43	0∞0∞1∞0101∞00∞∞1000∞0010100101∞∞0∞1111∞0∞∞0	89 (0.999742)	93
44	∞∞∞0∞0∞∞00∞010000∞11110001∞∞011∞01101∞0∞010∞	91 (0.999508)	93

Chapter 5

Conclusions

We have proposed techniques for constructing good covering arrays of strength two and three using group theory coupled with computer search. For $t = 2$, a comparison of our fixed symbols construction with heuristic tools like NIST IPOG-F shows that our construction provides significantly smaller size covering arrays. For $t = 3$, a comparison of our construction with best known covering arrays shows that our construction provides smaller testing arrays with good coverage measure. It is an interesting open problem to extend this algebraic construction to construct covering arrays of strength up to 6.

Bibliography

[1] Akthar, Y., Maity, S., Chandrasekharan, R.C., Covering arrays of strength four with budget constraints, Mathematical and Computational Sciences, *Narosa*, 106-114, 2015.

[2] Brereton, P., Budgen, D., Component-based systems: a classification of issues, *IEEE Computer*, 33(11), 54-62, 2000.

[3] Boroday, S.Y., Grunskii, I.S., Recursive generation of locally complete tests, *Cybernetics and Systems Analysis*, 28, 20-25, 1992.

[4] Carroll, C.T, The cost of poor testing: A U.S. government study (part 1). *ED-PACS, The EDP Audit, Control and Security Newsletter*, 31(1), 117, 2003.

[5] Chandrasekharan, R.C., Maity, S., Covering arrays of strength two and pairwise software testing, Mathematical and Computational Sciences, *Narosa*, 115-121, 2015.

[6] Chateauneuf, M.A., Colbourn, C.J., Kreher, D.L., Covering Arrays of Strength Three, *Designs, Codes and Cryptography*, 16, 235-242, 1999.

[7] Cohen, D.M., Dalal, S.R., Fredman, M.L., Patton, G. C., The AETG system: An Approach to Testing Based on Combinatorial Design, *IEEE Transactions on Software Engineering*, 23 (7), 437-443, 1997.

[8] Cohen, D M., Dalal, S.R., Parelius, J., Patton , G.C., The combinatorial design approach to automatic test generation, *IEEE Software*, 13, 83-88, 1996.

[9] Colbourn, C., Covering Array Tables for t=2,3,4,5,6 available at http://www.public.asu.edu/ ccolbou/src/tabby/catable.html

[10] Covering Arrays generated by IPOG-F, available at http://math.nist.gov/coveringarrays/ipof/ipof-results.html

[11] Edelman, A., The mathematics of the Pentium division bug, *SIAM Review*, 39, 54-67, 1997.

[12] Fisher, R. A., An examination of the different possible solutions of a problem in incomplete blocks, *Ann. Eugenics*, 10, 52-75, 1940.

[13] Hartman, A., Raskin, L., Problems and algorithms for covering arrays, *Discrete Mathematics*, 284, 149-156, 2004.

[14] Hartman, A., Software and hardware testing using combinatorial covering suites, in Graph Theory, Combinatorics and Algorithms, *Interdisciplinary Applications, Kluwer Academic Publishers*, 34, 237-266, 2006.

[15] Hedayat, A., Sloane, N., Stufken, J., Orthogonal Arrays, *Springer-Verlag*, New York, 1999.

[16] Kuhn, R.D., Wallace, D.R., Gallo, A.M., Software Fault Interactions and Implications for Software Testing, *IEEE Trans. on Software Engineering*, 30 (6), 418-421, 2004.

[17] Lei, Y., Tai, K.C., In-Parameter-Order: A Test Generating Strategy for Pairwise Testing, *IEEE Trans. on Software Engineering*, 28 (1), 1-3, 2002.

[18] Leveson, N., Turner, C.S., An investigation of the Therac-25 accidents, *IEEE Computer*, 26, 18-41, 1993.

[19] Lions, J.L., Ariane 5 Flight 501 failure, Report by the Inquiry board, http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html.

[20] Lobb, J.R., Coulbourn, C.J., Danziger, P., Stevens, B., Jose Torres-Jimenez, J., Cover starters for covering arrays of strength two, *Discrete Mathematics*, 312, 943-956, 2012.

[21] Maity, S., Nayak, A., Improved Test Generation Algorithms for Pair-Wise Testing, *Proc. 16th IEEE International Symposium on Software Reliability Engineering*, 235-244, 2005.

[22] Maity, S., Nayak, A., Zaman, M., Srivastava, A., Goel, N., An Efficient Test Generating Algorithm for Pair-wise Testing, *Proc. 14th Intl. Symposiam on Software Reliability Engineering*, 2003, Colorado, Denver, USA.

[23] Maity, S., 3-Way software testing with budget constraints, *IEICE Trans. on Information and Systems*, E95-D (9), 2227-2231, 2012.

[24] Meagher, K., Stevens, B., Group construction of covering arrays, *J. Combinatorial Designs*, 13, 70-77, 2005.

[25] Myers, G.J., The art of software testing, *John Wiley & Sons Inc*, 2004.

[26] Pairwise testing website: http://www.pairwise.org/.

[27] Robinson, D.J.S., A course in the theory of groups, Second Edition, *Springer*, 1995.

[28] Shasha, D.E., Kouranov, A.Y., Lejay, L.V., Chou, M.F., Coruzzi, G.M., Using Combinatorial Design to Study Regulation by Multiple Input Signals. A Tool for Parsimony in the Post-Genomics Era, *Plant Physiology*, 127, 1590-1594, 2001.

[29] Stevens, B., Mendelsohn, E., New recursive methods for transversal covers, *J. of Combinatorial Designs*, 7 (3), 185-203, 1999.

[30] Tang, D.T., Chen, C.L., Iterative exhaustive pattern generation for logic testing, *IBM J. Res. Develop*, 28, 212-219, 1984.

[31] Tang, D.T., Woo, L. S., Exhaustive test pattern generation with constant weight vectors, *IEEE Trans. Computers*, 32, 1145-1150, 1983.

[32] Tong, A.-I., Wung, Y.-G., Li, L.-D, Room-temperature phosphorimetry studies of some addictive drugs following dansyl chloride labelling, *Talanta*, 1429-1436, 1996.

[33] Vadde, K.K., Syrotiuk, V.R., Factor interaction on service delivery in mobile ad hoc networks, *IEEE Selected Areas in Communications*, 1335-1346, 2004.

[34] Weiss, G.W., Duping the Soviets: The Farewell Dossier, *The New York Times*, 2013.

[35] Weyuker, E.J, Testing component-based software: A cautionary tale, *IEEE Software*, 15 (5), 54-59, 1998.

[36] Yilmaz, C., Cohen, M., and Porter, A., Covering arrays for efficient fault characterisation in complex configuration spaces, *IEEE Trans. on Software Engineering*, 32 (1), 20-34, 2006.

Appendix A

Sample programs for strength 2

A.1 Program for g=3, k=9, f=1

```
#include<iostream>
using namespace std;

/*generate strings and check for the representation of orbits*/
int str(int *V,int *S,int l,int k)
{
 l++;int r;
 for (int i=0;i<2;i++)
  {V[l]=i;
   if(l==k)
    {l++;


      int n=0;
      for(int p=0;p<9;p++)
       {if (( p!=1))
          {S[p]=V[n];
           n++;
          }
       }
       int D[2];
       int a1,a2;
      int w=1;

      for(int p=1;p<8;p++)
       {a1=0;a2=0;
        for(int q=0;q<8;q++)
         {D[0]=S[q];D[1]=S[(q+p)%9];
          if((D[0] == 0 && D[1]==0) ||(D[0] == 1 && D[1]==1))
           a1=1;
          else
           if((D[0] == 0 && D[1]==1) ||(D[0] == 1 && D[1]==0))
            a2=1;
         }
        if(a1==1 && a2 ==1)
         w=1;
        else
         {w=0;break;}
       }

     if(w==1)
      {cout<<"starter: \t\t";
       for(int p=0;p<9;p++)
        cout<<S[p];
       cout<<"\n";

      }
    }
   if(l<k)
    r=str(V,S,l,k);
  }
 return(0);
}

int main()
{int S[9];//starter
```

```
 int *V;
 V = new int[8];

 for(int i=0;i<8;i++)
  V[i]=0;

 for(int i=0;i<9;i++)
  if( i== 1)
   S[i]=2;
  else
      S[i]=0;
 int r;
 r=str(V,S,-1,8);

 delete [] V;
 return(0);
}
```

A.2 Program for g=4, k=5, f=1

```
#include<iostream>
using namespace std;

/*generate vectors and check for representation of orbits*/
int str(int *V,int *S,int l,int k)
{
 l++;int r;
 for (int i=0;i<3;i++)
  {V[l]=i;
   if(l==k)
    {l++;

     int n=0;
     for(int p=0;p<5;p++)
```

```
{if ( p!=1)
   {S[p]=V[n];
    n++;
   }
}
int D[2];
int a1,a2,a3;
int w=1;

for(int p=1;p<4;p++)
 {a1=0;a2=0,a3=0;
   for(int q=0;q<5;q++)
    {D[0]=S[q];D[1]=S[(q+p)%5];
     if((D[0] == 0 && D[1]==0) ||(D[0] == 1 && D[1]==1) ||(D[0] == 2 &&
    D[1]==2) )
       a1=1;
     else
      if((D[0] == 0 && D[1]==1) ||(D[0] == 1 && D[1]==2)||(D[0] == 2 &&
     D[1]==0) )
        a2=1;
      else
       if((D[0] == 0 && D[1]==2) ||(D[0] == 1 && D[1]==0)||(D[0] == 2 &&
      D[1]==1) )
         a3=1;
    }
   if(a1==1 && a2 ==1 && a3==1)
    w=1;
   else
    {w=0;break;}
 }

if(w==1)
 {cout<<"\n\n";
 for(int p=0;p<5;p++)
```

```
          cout<<S[p];
         cout<<"\n";
        }
     }
    if(l<k)
     r=str(V,S,l,k);
  }
  return(0);
}

int main()
{int S[5];//starter
 int V[4];
 for(int i=0;i<4;i++)
  V[i]=0;

 for(int i=0;i<5;i++)
   if( i== 1)
    S[i]=3;
   else
    S[i]=0;
 int r;
 r=str(V,S,-1,4);
 return(0);
}
```

A.3 Program for g=4, k=11, f=2

```
#include<iostream>
using namespace std;

/*generate vectors and check for representation of orbits*/
int str(int *V,int *S,int l,int k)
{
 l++;int r;
```

```
for (int i=0;i<2;i++)
 {V[l]=i;
  if(l==k)
   {l++;

     int n=0;
     for(int p=0;p<11;p++)
      {if ((p!=1 && p!=3) && (p!=8 && p!=7))
        {S[p]=V[n];
         n++;
        }
      }
      int D[2];
      int a1,a2;
      int w=1;

      for(int p=1;p<10;p++)
       {a1=0;a2=0;
        for(int q=0;q<11;q++)
         {D[0]=S[q];D[1]=S[(q+p)%11];
          if((D[0] == 0 && D[1]==0) ||(D[0] == 1 && D[1]==1))
           a1=1;
          else
           if((D[0] == 0 && D[1]==1) ||(D[0] == 1 && D[1]==0))
            a2=1;
         }
        if(a1==1 && a2 ==1)
         w=1;
        else
         {w=0;break;}
       }

     if(w==1)
      {for(int p=0;p<11;p++)
```

```
            cout<<S[p];
          cout<<"\n";

        }
      }
    if(l<k)
      r=str(V,S,l,k);
  }
  return(0);
}

int main()
{int S[11];//starter
  int V[7];

  for(int i=0;i<7;i++)
   V[i]=0;

  for(int i=0;i<11;i++)
    if(i==1 || i== 3)
     S[i]=2;
    else
      if(i==8 || i==7)
       S[i]=3;
       else
        S[i]=0;
  int r;
  r=str(V,S,-1,7);
  return(0);
}
```

A.4 Program for g=5, k=9, f=2

```
#include<iostream>
using namespace std;
```

```
/*generate vectors and check for representation of orbits*/
int str(int *V,int *S,int l,int k)
{
 l++;int r;
 for (int i=0;i<3;i++)
  {V[l]=i;
   if(l==k)
    {l++;

     int n=0;
     for(int p=0;p<9;p++)
      {if ((p!=0 && p!=1) && (p!=3 && p!=7))
        {S[p]=V[n];
         n++;
        }
      }
      for(int p=0;p<9;p++)
       cout<<S[p];
      cout<<"\n";

      int D[2];
      int a1,a2,a3;
      int w=1;

      for(int p=1;p<8;p++)
       {a1=0;a2=0,a3=0;
         for(int q=0;q<9;q++)
          {D[0]=S[q];D[1]=S[(q+p)%9];
           if((D[0] == 0 && D[1]==0) ||(D[0] == 1 && D[1]==1) ||
           (D[0] == 2 && D[1]==2) )
            a1=1;
           else
            if((D[0] == 0 && D[1]==1) ||(D[0] == 1 && D[1]==2)||
                 (D[0] == 2 && D[1]==0) )
                  a2=1;
                 else
                  if((D[0] == 0 && D[1]==2) ||(D[0] == 1 && D[1]==0)||
                  (D[0] == 2 && D[1]==1) )
                   a3=1;
              }
             if(a1==1 && a2 ==1 && a3==1)
              w=1;
             else
              {w=0;break;}
           }

         if(w==1)
          {for(int p=0;p<9;p++)
             cout<<S[p];
           cout<<"\n";
          }
      }
    if(l<k)
      r=str(V,S,l,k);
  }
  return(0);
}

int main()
{int S[9];//starter
  int V[5];
  for(int i=0;i<5;i++)
   V[i]=0;

  for(int i=0;i<9;i++)
    if(i==0 || i== 1)
     S[i]=3;
  else
   if(i==3 || i==7)
    S[i]=4;
    else
     S[i]=0;
 int r;
 r=str(V,S,-1,5);
 return(0);
}
```

A.5 Program for g=6, k=9, f=1

```
#include<iostream>
using namespace std;

/*generate vectors and check for representation of orbits*/
int str(int *V,int *S,int l,int k)
{
 l++;int r;
 for (int i=0;i<5;i++)
  {V[l]=i;
   if(l==k)
    {l++;

     int n=0;
     for(int p=0;p<9;p++)
      {if ( p!=1)
        {S[p]=V[n];
         n++;
        }
      }
      int D[2];
      int a1,a2,a3,a4,a5,a,b;
      int w=1;
```

```
for(int p=1;p<9;p++)
 {a1=0;a2=0,a3=0;a4=0;a5=0;
  for(int q=0;q<9;q++)
   {D[0]=S[q];D[1]=S[(q+p)%9];
    if(D[0]==5 || D[1]==1)
     {a1=1;a2=1,a3=1;a4=1,a5=1;
     }
     else
     {if(D[0] > D[1] )
     {a=(6-D[0])+D[1];}
    else
     {a=D[1]-D[0];}
    if(a%6==0)
     a1=1;
    else
     if(a%6==1 )
      a2=1;
     else
      if(a%6==2 )
       a3=1;
      else
       if(a%6==3 )
        a4=1;
       else
        if(a%6==4)
         a5=1;
    }
   }
  if(a1==1 && a2 ==1 && a3==1 && a4==1 && a5==1)
   w=1;
  else
   {w=0;break;}
 }
```

```
        if(w==1)
         {cout<<"\n";
         for(int p=0;p<9;p++)
            cout<<S[p];
          cout<<"\n";

          }
     }
   if(l<k)
    r=str(V,S,l,k);
  }
  return(0);
}

int main()
{int S[9];//starter
  int V[8];
  for(int i=0;i<8;i++)
   V[i]=0;

  for(int i=0;i<9;i++)
   if( i== 1)
    S[i]=5;
   else
    S[i]=0;
  int r;
  r=str(V,S,-1,8);
  return(0);
}
```

A.6 Program for g=7, k=17, f=3

```
#include<iostream>
using namespace std;
```

```
/*generate vectors and check for representation of orbits*/
int str(int *V,int *S,int l,int k)
{
 l++;int r;
 for (int i=0;i<4;i++)
  {V[l]=i;
   if(l==k)
    {l++;

     int n=0;
     for(int p=0;p<17;p++)
      {if (( p!=10 && p!=4) && (p!=11 && p!=7) && (p!=12 &&
      p!=9 && p!=2))
        {S[p]=V[n];
         n++;
        }
      }
      int D[2];
      int a1,a2,a3,a4,a;
      int w=1;

      for(int p=1;p<17;p++)
       {a1=0;a2=0,a3=0;a4=0;
        for(int q=0;q<17;q++)
         {D[0]=S[q];D[1]=S[(q+p)%17];
          if(D[0]==6|| D[1]==6 || D[0]==5 || D[1]==5 ||
          D[0]==4 || D[1]==4)
           {;}
           else
           {if(D[0] > D[1] )
           {a=(4-D[0])+D[1];}
          else
           {a=D[1]-D[0];}
          if(a%4==0)
```

```
              a1=1;
             else
              if(a%4==1  )
               a2=1;
              else
               if(a%4==2  )
                a3=1;
               else
                if(a%4==3  )
                 a4=1;
               }
           }
          if(a1==1 && a2  ==1 && a3==1 && a4==1)
           w=1;
          else
           {w=0;break;}
         }

        if(w==1)
         {cout<<"\n";
         for(int  p=0;p<17;p++)
           cout<<S[p];
          cout<<"\n";
        }
      }
    if(l<k)
     r=str(V,S,l,k);
  }
  return(0);
}

int  main()
{int  S[17];//starter
 int  V[10];
 for(int i=0;i<10;i++)
  V[i]=0;

 for(int i=0;i<17;i++)
  if(i==10 || i==4)
   S[i]=4;
  else if(i==11 || i==7)
   S[i]=5;
  else if(i== 12 || i==2 || i==9)
   S[i]=6;
  else
   S[i]=0;
 int r;
 r=str(V,S,-1,10);
 return(0);
}
```

A.7 Program for g=8, k=18, f=3

```
#include<iostream>
using namespace std;

/*generate vectors and check for representation of orbits*/
int str(int *V,int *S,int l,int k)
{
 l++;int r;
 for (int i=0;i<5;i++)
  {V[l]=i;
   if(l==k)
    {l++;


     int n=0;
     for(int p=0;p<18;p++)
      {if (( p!=10 && p!=15) && (p!=11 && p!=8) && (p!=17 && p!=9))
        {S[p]=V[n];
   n++;
  }
}
int D[2];
int a1,a2,a3,a4,a5,a;
int w=1;

for(int p=1;p<18;p++)
 {a1=0;a2=0,a3=0;a4=0;a5=0;
  for(int q=0;q<18;q++)
   {D[0]=S[q];D[1]=S[(q+p)%18];
    if(D[0]==6|| D[1]==6 || D[0]==5 || D[1]==5 || D[0]==7 || D[1]==7)
     {;}
     else
     {if(D[0] > D[1] )
     {a=(5-D[0])+D[1];}
    else
     {a=D[1]-D[0];}
    if(a%5==0)
     a1=1;
    else
     if(a%5==1 )
      a2=1;
     else
      if(a%5==2 )
       a3=1;
      else
       if(a%5==3 )
        a4=1;
       else
        if(a%5==4)
         a5=1;
      }
   }
```

```
            if(a1==1 && a2 ==1 && a3==1 && a4==1 && a5==1)
             w=1;
            else
             {w=0;break;}
           }

        if(w==1)
         {cout<<"\n";
         for(int p=0;p<18;p++)
            cout<<S[p];
           cout<<"\n";

         }
     }
    if(l<k)
     r=str(V,S,l,k);
  }
  return(0);
}

int main()
{int S[18];//starter
int V[12];

  for(int i=0;i<12;i++)
   V[i]=0;

  for(int i=0;i<18;i++)
    if(i==10 || i==15)
     S[i]=5;
    else if(i==11 || i==8)
     S[i]=6;
    else if(i== 9 || i==17)
     S[i]=7;
    else
      S[i]=0;
  int r;
  r=str(V,S,-1,12);
  return(0);
}
```

Appendix B

Sample programs for strength 3

B.1 Program for g=3, k=16, f=1

```
#include<iostream>
#include<math.h>

using namespace std;

/*generate vectors and check for representation of orbits*/
int str(int *V,int *S,int *n,int *A,int l,int rk,int k,float &M)
{if(M==1)
   return(0);
  else
  {l++;int r;
  for (int i=0;i<2;i++)
   {V[l]=i;
    if(l==rk)
     {l++;


       int nn=0;
       for(int p=0;p<k;p++)
        {if ( p==1 || p==2 ||p==4 || p==5 || p==7 || p==12||
        p==13 ||p==14  )
            {;}
    else
     {S[p]=V[nn];
      nn++;
     }
   }
   int D[3];
   int w=1;int c;
   D[0]=0;D[1]=0;D[2]=0;
   float m=0.0; int cc;cc=0;
  for(int x=1;x<k;x++)
   {for(int y=x;y<k;y++)
     if((x+y)<=k-2 && ((k+(k-y)+(k-x))%k > x))
     {
      for(int q=0;q<14;q++)
       A[q]=0;
      A[0]=1;A[13]=1;
      cc++;
      for(int q=0;q<k;q++)
       {D[0]=S[q];D[1]=S[(q+x)%k];D[2]=S[(q+x+y)%k];
        if((D[0] == 0 && D[1]==0 && D[2]==1 ) ||(D[0] == 1
        && D[1]==1 && D[2]==0) )
         A[1]=1;
        else
         if((D[0] == 0 && D[1]==0 && D[2]==2 ) ||(D[0] == 1
         && D[1]==1 && D[2]==2) )
          A[2]=1;
         else
          if((D[0] == 0 && D[1]==1 && D[2]==0 ) ||(D[0] == 1
          && D[1]==0 && D[2]==1) )
           A[3]=1;
          else
           if((D[0] == 0 && D[1]==1 && D[2]==1 ) ||(D[0] == 1
           && D[1]==0 && D[2]==0) )
           A[4]=1;
          else
           if((D[0] == 0 && D[1]==1 && D[2]==2 ) ||(D[0] == 1
          && D[1]==0 && D[2]==2) )
          A[5]=1;
            else
             if((D[0] == 0 && D[1]==2 && D[2]==0 ) ||(D[0] == 1
            && D[1]==2 && D[2]==1)   )
             A[6]=1;
              else
               if((D[0] == 0 && D[1]==2 && D[2]==1 ) ||(D[0] == 1
              && D[1]==2 && D[2]==0) )
               A[7]=1;
                else
                 if((D[0] == 0 && D[1]==2 && D[2]==2 ) ||(D[0] == 1
                && D[1]==2 && D[2]==2)   )
                 A[8]=1;
                  else
                   if  ((D[0] == 2 && D[1]==0 && D[2]==0 ) ||(D[0] == 2
                  && D[1]==1 && D[2]==1))
                   A[9]=1;
                    else
                     if((D[0] == 2 && D[1]==0 && D[2]==1 ) ||(D[0] == 2
                    && D[1]==1 && D[2]==0) )
                     A[10]=1;
                      else
                       if((D[0] == 2 && D[1]==0 && D[2]==2 ) ||(D[0] == 2
                      && D[1]==1 && D[2]==2) )
                       A[11]=1;
                        else
                         if((D[0] == 2 && D[1]==2 && D[2]==0 ) ||(D[0] == 2
                        && D[1]==2 && D[2]==1) )
                         A[12]=1;
  }
 c=0;
```

```
                for(int w=0;w<14;w++)
                  if(A[w]==0)
                    c=c+n[w];

                 m=m+16*(pow(3,3)-c);

              }
            }
          m=m/15120.0;
          if(m>M)
            {M=m;
              cout<<"M="<<M<<" ␣";
              for(int p=0;p<k;p++)
                cout<<S[p];
              cout<<"\n"<<flush;



        }
      }
    if(l<rk)
      r=str(V,S,n,A,l,rk,k,M);
  }}

}



float M;
int main()
{int f,k;

  f=1;M=0.0;
  k=16;
  int S[16];//starter
  int V[8];
```

```
 int n[14];
 int A[14];
 for(int i=0;i<13;i++)
  n[i]=2;
 n[13]=1;

 for(int i=0;i<8;i++)
  V[i]=0;
 for(int i=0;i<16;i++)
  S[i]=0;

 for(int p=0;p<16;p++)
  if( p==1 || p==2 ||p==4 || p==5 || p==7 || p==12|| p==13 ||p==14 )
   S[p]=2;
  else
   S[p]=0;
 int r;
 r=str(V,S,n,A,-1,8,16,M);

return(0);

}
```

B.2 Program for g=3, k=30, f=1

```
#include<iostream>
#include<math.h>

using namespace std;

/*generate vectors and check for representation of orbits*/
int str(int *V,int *S,int *n,int *A,int l,int rk,int k,float &M)
{if(M==1)
  return(0);
 else
```

```
{l++;int r;
for (int i=0;i<2;i++)
 {V[l]=i;
  if(l==rk)
   {l++;


    int nn=0;
    for(int p=0;p<k;p++)
     {if (p==1 || p==2 ||p==5|| p==12|| p==13|| p==19|| p==22 ||p==25
     || p==27 || p==29 )
       {;}
      else
       {S[p]=V[nn];
        nn++;
       }
     }
     int D[3];int cc;cc=0;
     int w=1;int c;
     D[0]=0;D[1]=0;D[2]=0;
    float m=0.0;
    for(int x=1;x<k;x++)
     {for(int y=x;y<k;y++)
       if((((x+y)<=k-2) && ((k+(k-y)+(k-x))%k > x)) || (k%3==0 && x==k/3
       && y== k/3 && (k+(k-y)+(k-x))%k==k/3))
       {
        for(int q=0;q<14;q++)
         A[q]=0;
        A[0]=1;A[13]=1;
        cc++;
        for(int q=0;q<k;q++)
         {D[0]=S[q];D[1]=S[(q+x)%k];D[2]=S[(q+x+y)%k];

          if((D[0] == 0 && D[1]==0 && D[2]==1 ) ||(D[0] == 1 &&
         D[1]==1 && D[2]==0) )
 A[1]=1;
else
 if((D[0] == 0 && D[1]==0 && D[2]==2 ) ||(D[0] == 1 &&
 D[1]==1 && D[2]==2) )
  A[2]=1;
 else
  if((D[0] == 0 && D[1]==1 && D[2]==0 ) ||(D[0] == 1 &&
  D[1]==0 && D[2]==1) )
   A[3]=1;
  else
   if((D[0] == 0 && D[1]==1 && D[2]==1 ) ||(D[0] == 1 &&
   D[1]==0 && D[2]==0) )
   A[4]=1;
   else
    if((D[0] == 0 && D[1]==1 && D[2]==2 ) ||(D[0] == 1 &&
    D[1]==0 && D[2]==2) )
   A[5]=1;
    else
     if((D[0] == 0 && D[1]==2 && D[2]==0 ) ||(D[0] == 1
     && D[1]==2 && D[2]==1)   )
      A[6]=1;
     else
      if((D[0] == 0 && D[1]==2 && D[2]==1 ) ||(D[0] == 1
      && D[1]==2 && D[2]==0) )
       A[7]=1;
      else
       if((D[0] == 0 && D[1]==2 && D[2]==2 ) ||(D[0] == 1
       && D[1]==2 && D[2]==2)   )
        A[8]=1;
       else
        if  ((D[0] == 2 && D[1]==0 && D[2]==0 ) ||(D[0] == 2
        && D[1]==1 && D[2]==1))
         A[9]=1;
        else
```

```
                    if((D[0] == 2 && D[1]==0 && D[2]==1 ) ||(D[0] == 2
                    && D[1]==1 && D[2]==0) )
                     A[10]=1;
                    else
                     if((D[0] == 2 && D[1]==0 && D[2]==2 ) ||(D[0] == 2
                     && D[1]==1 && D[2]==2) )
                      A[11]=1;
                     else
                      if((D[0] == 2 && D[1]==2 && D[2]==0 ) ||(D[0] == 2
                      && D[1]==2 && D[2]==1) )
                       A[12]=1;
         }
        c=0;
        for(int w=0;w<14;w++)
         if(A[w]==0)
          c=c+n[w];

         if(k%3==0 && x==k/3 && y== k/3 && (k+(k-y)+(k-x))%k==k/3)
          m=m+(k/3)*(pow(3,3)-c);
         else
          m=m+k*(pow(3,3)-c);

    }
   }
  m=m/(((k*(k-1)*(k-2))/6.0)*27.0);

  if(m>M)
   {M=m;
    cout<<"M="<<M<<" ";
    for(int p=0;p<k;p++)
     cout<<S[p];
    cout<<"\n"<<flush;
```

```
      }
    }
   if(l<rk)
     r=str(V,S,n,A,l,rk,k,M);
 }}

}



float M;
int main()
{int f,k;
 int n[14];
 int A[14];
 for(int i=0;i<13;i++)
   n[i]=2;
 n[13]=1;



 f=1;M=0.0;
 k=30;
 int S[30];
 int V[20];



 for(int i=0;i<20;i++)
   V[i]=0;
 for(int i=0;i<30;i++)
   S[i]=0;

 for(int p=0;p<30;p++)
   if(p==1 || p==2 ||p==5|| p==12|| p==13|| p==19|| p==22 ||p==25
   || p==27 || p==29 )
     S[p]=2;
    else
      S[p]=0;
  int r;
  r=str(V,S,n,A,-1,20,30,M);

return(0);

}
```