Group Construction of Covering Arrays

Ruchi Gupta

Dissertation
Submitted in Partial Fulfillment
Final year Project
5 Year BS-MS Dual Degree Program

Thesis Advisory Committee
Supervisor
Dr.Soumen Maity

Department of Mathematics
Indian Institute of Science Education and Research Pune

April 2011

Certificate

This is to certify that Ruchi Gupta, has carried out the work in this report titled "**Group Construction of Covering Arrays**" under my supervision and that it has not been submitted elsewhere for the award of any degree.

Signature of student
Name : Ruchi Gupta

Signature of Supervisor
Name : Dr. Soumen Maity

Acknowledgements

I would like to thank my project guide **Dr. Soumen Maity** for his continuous help in my project. He always listened to me and gave his advice to clarify my doubts.

Abstract

There has been a lot of research on covering arrays in the last two decades and most of it includes construction, application and generalization of covering arrays. The main focus of this thesis is the group construction of covering arrays. The project was aimed at improving bounds on covering array number and thus to improve applications of covering arrays to testing systems and networks. For this purpose, we use group construction of covering arrays which has been proven to be one of the best possible method for the construction of covering arrays.

A covering array $CA(n, k, g)$, is an $k \times n$ array with entries from Z_g and the property that in any pair of rows, each of the g^2 ordered pairs from $Z_g \times Z_g$ appear in at least one column. This property is called *qualitative independence*. Here, the size of the covering array is represented by parameter n and Z_g represents its alphabet. A covering array is *optimal* if it has the minimum number of columns among covering arrays with the same number of rows.

We use group construction of covering arrays with some modifications which yields new upper bounds on the size of the optimal covering arrays.

Contents

Chapter 1

Introduction

In the last few decades, covering arrays has been the focus of most of the research. They are actually termed as the natural generalization of orthogonal arrays. Covering arrays are computationally difficult to find and are of multiple applications, particularly software testing purposes [1].

Testing is very important but generally very expensive part of the development process of hardware and software. Out of the many possible configurations of software, each could be faulty. As mentioned above testing a large software or hardware system is a labor-intensive process and it also requires a huge amount of time and resources. The main purpose of software testing is to make sure that no actual error occurs in any such configuration. In order to achieve this goal, one can simulate all possible configurations and check that no error occurs but since there are too many possible configurations to check, this method is practically impossible. It can be realized that testing all configurations of a software or a hardware can be redundant and that it is possible to achieve a fair simulation of all of the program's behavior with a smaller number of chosen configurations [2].

Often inadequate testing results in major disasters, some examples for that are the software failure in the Therac-5 radiation therapy machine that resulted in deaths and severe injuries to cancer patients due to overdoses of radiation [3]. Also, due to inadequate testing another extreme catastrophic disaster occurred when the Ariane 5 satellite launcher exploded in 40 seconds into its maiden flight, due to this overflow failure both processors had to shut down resulting in aborting the satellite mission [4]. Intel corporation had to suffer millions of dollars due to the Pentium floating point division bug that caused error in accuracy of only small number of division computations [5].

Most of the work on covering arrays focuses on developing constructions for them and finding bounds on their minimum size. A covering array $CA(n, k, g)$, is an $k \times n$ array with entries from Z_g and the property that in any pair of rows, each of the g^2 ordered pairs from $Z_g \times Z_g$ appear in at least one column. This property is called *qualitative independence*. The parameter n represents the size of the covering array

and Z_g is its alphabet.

It is often desirable to have covering array with smallest possible size i.e. covering array with minimum number of columns possible. The main focus of the thesis is *group construction of covering array* is presented in Chapter 3 with slight modifications and the tables in chapter 4 lists the bounds that are found with this modified group construction techniques. The group construction produces some of the smallest known covering arrays for specific parameters.

1.1 The History

Covering array are also known as Qualitatively independent families. They are generalization of well-studied orthogonal arrays [6]. Orthogonal Array which are first developed by Rao in 1940s [7] although provide good test designs, but for some parameters it is not possible to build an orthogonal array so relaxation of orthogonal arrays, called covering arrays are developed [8]. Most research is about developing new and better methods to find the bounds on the minimum sizes of the covering array. For binary covering array ($k = 2$) the exact size of minimum covering array is known, but as k increases the task becomes much more difficult [9].

1.2 Applications of Covering Arrays

There are often variety of applications of covering arrays including correct functioning of radiation therapy machines, satellite launching and many more. So, to lower the cost of software testing is crucial. One approach to lower the cost of software testing was suggested by Cohen, Dalal, Fredman and Patton [10], which is explained below with the help of the following example [11].

Suppose an internet site has to be tested to function correctly, the system has four parameters : operating systems, browsers, printers and communication protocols. The first parameter, operating system has three values i.e Windows, Linux and Solaris. The second parameter, browser, has two values i.e. Explorer and Netscape. The third parameter has three values and the fourth parameter has two values i.e. Epson, HP, IBM and Token Ring and Ethernet respectively.

In total we have, $3{\times}2 \times 3 \times 2 = 36$ possible test configurations, but if following nine tests are done, they cover all pairwise interactions between different parameters of the system.

Operating system	Browser	Printer	Protocol
Windows	Explorer	Epson	Token Ring
Windows	Netscape	HP	Ethernet
Windows	Explorer	IBM	Ethernet
Linux	Netscape	Epson	Token Ring
Linux	Explorer	HP	Ethernet
Linux	Netscape	IBM	Token Ring
Solaris	Explorer	Epson	Ethernet
Solaris	Netscape	HP	Token Ring
Solaris	Explorer	IBM	Ethernet

In this table, the interaction between operating system and printers are covered only once, but interactions between operating systems and browsers are covered more than once (twice).
As we can see clearly from the table that Linux and Netscape, Solaris and Explorer and Windows and Explorer are tested together twice in the test suite.

Chapter 2

Covering Arrays and Related Designs

Covering Arrays are mathematically very relevant designs in terms of their multiple applications. They are the generalization of well studied and well known orthogonal arrays.
In this chapter, we will give the overview of all the mathematical designs that are relevant in our study of covering arrays including orthogonal arrays, latin squares and transversal designs.

2.1 Mutually Orthogonal Latin Squares (MOLS)

Latin Squares are first developed and studied by Euler in 1782. They are very old but relevant designs in terms of their applications. They are quite closely related to orthogonal arrays.

Definition of Latin Squares : An $n \times n$ array (n being a positive integer) on n symbols with the property that each symbol occurs exactly once in each row and each column is said to be a latin square of order n. For a latin square to be in its *standard form* the symbols in the first row should come in their natural order.

Example : Given below is the latin square of order 5 in its standard form. This is also the addition table for Z_5.

$$\mathrm{L} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 & 0 \\ 2 & 3 & 4 & 0 & 1 \\ 3 & 4 & 0 & 1 & 2 \\ 4 & 0 & 1 & 2 & 3 \end{pmatrix}$$

Remark : A latin square of order n exists for every n, i.e. an addition table for Z_n is always a latin square of order n. If we permute the rows or the columns of a latin square, we will still get a latin square.

Definition of Orthogonal Latin Squares : Let n be a positive integer and let A = $[a_{ij}]$ and B = $[b_{ij}]$ be latin squares of order n. Now, if in the $n \times n$ array with entries $[(a_{ij},b_{ij})]$, each of the n^2 ordered pairs of symbols occurs exactly once, then A and B are orthogonal latin squares.

Example : Given below are two orthogonal latin squares of order 3 (both in their standard form).

$$L_1 = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{pmatrix} \; L_2 = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{pmatrix}$$

Now, consider the $n \times n$ array with entries $[(l_{1_{i,j}},l_{2_{i,j}})]$

$$[L_1, L_2] = \begin{pmatrix} (0,0) & (1,1) & (2,2) \\ (2,1) & (0,2) & (1,0) \\ (1,2) & (2,0) & (0,1) \end{pmatrix}$$

Since, in this array each of the 9 ordered pairs occurs exactly once, therefore L_1 and L_2 are orthogonal latin Squares.

Remark : A set of latin squares with the property that any two distinct squares from the set are orthogonal is called a set of mutually orthogonal latin squares(MOLS). Let the cardinality of the largest set of MOLS of order n be $N(n)$. For all n,

$$1 \leq N(n) \leq n-1$$

A complete set of MOLS is a set of $n-1$ MOLS of order n [8]. The main point here is that if n is prime or prime power then $N(n) = n-1$.

2.2 Transversal Designs

Transversal Designs are also very relevant designs in the study of orthogonal arrays as there is a close relation between these two. They are the generalization of mutually orthogonal latin squares(MOLS).

Definition : A transversal design T(k, λ, g) (where k,λ,g be positive integers) is a triple (X,$\mathcal{G}$,$\mathcal{B}$) with the below properties:

1. A set of kg symbols, called varieties is X.
2. A partition set of the set X, $\mathcal{G} = G_1$,G_2,...,G_k, here each G_i is a $g - subset$ of the set X. These sets G_i are called groups.
3. A collection of $k - sets$ of X, $\mathcal{B}$, called blocks, with the property that each group is intersected by each block G_i, for $i = 1, ...k$ in exactly one variety.

4. Any pair of varieties from different groups occur in the same number of blocks. The number is called the index and is denoted by λ.

Example : Below given is a transversal design T[3, 1, 2] :

$$X = \{0, a, 1, b, 2, c\}$$

$$\mathcal{G} = \begin{pmatrix} G_1 = (0, 1) \\ G_2 = (a, b) \\ G_3 = (2, c) \end{pmatrix}$$

$$\mathcal{B} = \begin{pmatrix} B_1 = \{0, a, 2\} \\ B_2 = \{0, b, c\} \\ B_3 = \{1, a, c\} \\ B_4 = \{1, b, 2\} \end{pmatrix}$$

2.3 Orthogonal Arrays

Definition : An orthogonal array, $OA(n, k, g, t)$ (where n, k, g, t are positive integers and $t \leq k$) of index λ, with strength t and size g, is an $k \times n$ array with entries from $\{0, 1, ..., g-1\}$ and the property that any $t \times n$ subarray has all g^t possible $t - tuples$ occurring as columns exactly λ times.

Remark : In any $OA(n, k, g, t)$, $\lambda = n/g^t$.
If $t = 2$ i.e $\lambda = 1$, then the definition of orthogonal array goes like:
An orthogonal array is a $k \times g^2$ array with entries from Z_g such that within any pair of rows any ordered pairs from Z_g occurs in exactly one column.
A strength-2 $OA(n, k, g, 2)$ is equivalent to a transversal design T[k,λ,g] [6]. This equivalence is illustrated in the following example.

The following orthogonal array $OA(4, 3, 2, 2)$ is equivalent to the transversal design T[3,1,2] shown above :

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

The equivalence between $OA(4, 3, 2, 2)$ and T[3,1,2] can be explained in the following way :

For each row r_i of the $OA(4, 3, 2, 2)$ where $i \in$ {0,1,...,$k-1$} i.e $i \in$ {0,1,2} rewrite each letter a in the row as a_i i.e.

$$OA = \begin{pmatrix} 0_0 & 0_0 & 1_0 & 1_0 \\ 0_1 & 1_1 & 0_1 & 1_1 \\ 0_2 & 1_2 & 1_2 & 0_2 \end{pmatrix}$$

Now, the $kg = 6$ varieties in the transversal design T[3,1,2] are a_i for $a \epsilon$ {0,1,...,$g-1$} and $i \epsilon$ {0,1,...,$k-1$} i.e the 6 varieties in the transversal design are $\{0_0, 1_0, 0_1, 1_1, 0_2, 1_2\}$. Then, define the $k = 3$ groups of the transversal design by $G_i = \{a_i : a = 0, 1..., g-1\}$ for $i \epsilon$ {0,1,...,$k-1$} and the blocks in the transversal design are the columns in the covering array. Similarly, it is possible to construct an $OA(g^2, k, g, 2)$ from a $T[k, 1, g]$.

The equivalence between orthogonal arrays, transversal designs and MOLS are summarized in a theorem which is described below:

THEOREM *[12]*: *Let r,k be positive integers with k≥ 2 and r≥ 3. Then the existence of any one of the following designs implies the existence of the other two designs*:

1. an $OA(g^2, k, g, 2)$
2. a T$[k; 1; g]$
3. $k - 2$ $MOLS$ (g)

2.4 Covering Arrays

For certain values of k, it is impossible to build an orthogonal array (provided values of n, g, t are fixed) [13], increasing λ can be a solution to this problem, but for many practical purposes value of λ cannot be very large. Hence increasing λ does not offer any solution to our problem. If we make certain relaxations in the conditions of orthogonal arrays, we get what are known as covering arrays. Below is the definition:

Definition : A covering array t-$CA(n, k, g)$(where n, k, g, t are positive integers with $t \leq k$) with strength t and alphabet size g is an $k \times n$ array with entries from $\{0, 1, ..., g-1\}$ and the property that any $t \times n$ subarray has all g^t possible t-tuples occurring at least once.

Example : The below given array is a covering array 2-$CA(11, 5, 3)$:

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

An orthogonal array with 18 columns can be constructed using the above values $k = 5$, g =3 but it will have 7 more columns so it is beneficial to construct a covering array instead of orthogonal array.

Remark : The condition on orthogonal array that is relaxed in case of covering array is that each $t - tuple$ occurs exactly once is relaxed to be that each $t - tuple$ occurs at least once.
The number n, is the number of columns of the covering array t-$CA(n, k, g)$ is called the size of the covering array or the covering array number. The main focus of the research is to minimize this number as for many applications, CA with smallest possible size are used. The smallest possible size of the covering array is denoted by t-$CAN(k, g)$.

2.5 Qualitatively Independent Vectors

Definition : Two vectors $u, v \epsilon Z_g^n$ (where g, n are positive integers) are qualitatively independent if for each one of the possible g^2 ordered pairs $(a, b) \epsilon Z_g \times Z_g$, there is an index i so that $(u_i,v_i) = (a, b)$.

Example : Suppose g =3 and $n = 9$, so we have Z_3^9, therefore the possible $g^2 = 9$ ordered pairs are the following:

$\{(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)\}$, for $i = 2$, (u_2,v_2) = (0,1)

Remark : A set of vectors is qualitatively independent if any two distinct vectors in the set are qualitatively independent.
A set of rows in a covering array 2-$CA(n, k, g)$ is a set of k pairwise qualitatively independent vectors from Z_g^n.

Chapter 3

Constructions of Covering Arrays

Most of the work on covering arrays has focused on construction of covering arrays with minimum possible number of columns (which is a difficult problem) [14]. The constructions illustrated here for covering arrays give an upper bound for t-$CAN(k, g)$. We provide here three constructions namely : the finite field construction; the block size recursive construction and the group construction. Our main focus in this project is the group construction of the covering arrays. This construction has improved many upper bounds on covering array number.

3.1 Finite Field Construction

The construction given here is called finite field construction for orthogonal arrays [8]. This construction is used to construct a complete set of $MOLS$ of order k where k is a prime power. The construction builds an orthogonal array $OA(k^2, k+1, k, 2)$ and this is also a covering array $CA(k^2, k+1, k)$. The construction shown below is done through a series of lemmas and corollaries.

Lemma : Let k be a prime power, then $CAN(k+1, k) = k^2$.

Proof : Consider an array $C = (c_{ij})$ with $k+1$ rows and k^2 columns. In this proof we will index the rows and columns of C starting from 0. Consider $GF[k]$ to be a finite field of order k with the ordering of the elements as $\{f_0, f_1, f_2, ..., f_{k-1}\}$ and $f_0 = 0$ and $f_1 = 1$.
Let the first row of C be each element in the field repeated k times in the fixed order, so that the entry in column p of the first row is f_l where $l = \lfloor p/k \rfloor$. Now, for $i = \{0,1,2,...,k-1\}$, set the entry in the row $i+1$ and column p of the array C to be $f_i f_l + f_j$ where $l = \lfloor p/k \rfloor$ and $j \equiv p(mod k)$.
Now we need to see that any row of C is qualitatively independent from any other row of C.
First we will check that row 1 is qualitatively independent from all the other rows. We will prove this by contradiction.
Suppose row 1 and row n are not qualitatively independent. This means that for two distinct columns p and q, a pair is repeated between them. Now as there are only k_2

columns, if their is a repetition of any pair, then there will be problem in covering all possible pairs (x,y) such that $x,y \epsilon \{f_0,f_1,...,f_{k-1}\}$. According to condition discussed above, we have

$$(c_{1p}, c_{np}) = (c_{1q}, c_{nq})$$

$$f_{\lfloor p/k \rfloor} = f_{\lfloor q/k \rfloor} \tag{3.1}$$

$$f_{n-1}f_{\lfloor p/k \rfloor} + f_{j_1} = f_{n-1}f_{\lfloor q/k \rfloor} + f_{j_2} \tag{3.2}$$

where $j_1 = x \ mod \ k$ and $j_2 = y \ mod \ k$. From Equation 3.1, we have

$$\lfloor p/k \rfloor = \lfloor q/k \rfloor \tag{3.3}$$

Also from Equation 3.2, we have

$$f_{j_1} = f_{j_2} \tag{3.4}$$

i.e.

$$j_1 = j_2$$

this implies,

$$p = q(modk) \tag{3.5}$$

But Equations 3.2 and 3.5 cannot hold true simultaneously if $p \neq q$. Therefore, we arrive at contradiction and hence row 1 and any other row of C are qualitatively independent.
Now, we want to show that any two arbitrary rows are qualitatively independent. To prove this we again go by contradiction.
Assume that any two rows a and b are not qualitatively independent. Then for some distinct columns p and q, a pair is repeated between them.

$$(c_{ap}, c_{bp}) = (c_{aq}, c_{bq}) \tag{3.6}$$

thus,

$$f_{a-1}f_{\lfloor p/k \rfloor} + f_{j_3} = f_{a-1}f_{\lfloor q/k \rfloor} + f_{j_4} \tag{3.7}$$

and

$$f_{b-1}f_{\lfloor p/k \rfloor} + f_{j_3} = f_{b-1}f_{\lfloor q/k \rfloor} + f_{j_4} \tag{3.8}$$

where $j_3 = x \ mod \ k$ and $j_4 = y \ mod \ k$. Now, we subtract Equation 3.8 from 3.7, we get

$$f_{\lfloor p/k \rfloor}(f_{a-1} - f_{b-1}) = f_{\lfloor q/k \rfloor}(f_{a-1} - f_{b-1}) \tag{3.9}$$

which implies

$$\lfloor p/k \rfloor = \lfloor q/k \rfloor (\because a \neq b, f_{a-1} \neq f_{b-1}) \tag{3.10}$$

so we get,

$$f_{j_3} = f_{j_4} \tag{3.11}$$

$$j_3 = j_4$$

or,

$$x = y(mod k) \tag{3.12}$$

But Equations 3.12 and 3.10 cannot be true simultaneously if $x \neq y$. Therefore, we get a contradiction. Hence, rows a and b are qualitatively independent. So, we can say that all rows are qualitatively independent with all the other rows. By this method, we have constructed a $CA(k^2, k+1, k)$ which proves that $CAN(k+1, k) = k^2$ for k a prime power.

We show an example below of finite field construction a $CA(16, 5, 4)$ from the finite field construction from the finite field of four elements {0,1,2,3}

$$CA(16,5,4) = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \\ 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 1 & 0 & 3 & 2 & 2 & 3 & 0 & 1 & 3 & 2 & 1 & 0 \\ 0 & 1 & 2 & 3 & 2 & 3 & 0 & 1 & 3 & 2 & 1 & 0 & 1 & 0 & 3 & 2 \\ 0 & 1 & 2 & 3 & 3 & 2 & 1 & 0 & 1 & 0 & 3 & 2 & 2 & 3 & 0 & 1 \end{pmatrix}$$

Definition : Two *disjoint columns* in a covering array are the ones those have different entries for each row.

Corollary [8] : A covering array $CA(k^2, k, k)$ exists for any prime power k with the property that it has k disjoint columns .

Proof : Let the Covering array $CA(k^2, k+1, k)$ built by finite field construction be denoted by C. Let C' be the Covering Array formed by deleting the first row of C. Now, by finite field construction, for columns $j = \{0,1,2,...,k-1\}$, the entry on row i of C' is $f_i \times 0 + f_j = f_j$. Thus the first k columns are disjoint.

Example : Below is the example shown of $CA(16, 4, 4)$ with its first four columns disjoint.

$$CA(16,4,4) = \begin{pmatrix} 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 & 1 & 0 & 3 & 2 & 2 & 3 & 0 & 1 & 3 & 2 & 1 & 0 \\ 0 & 1 & 2 & 3 & 2 & 3 & 0 & 1 & 3 & 2 & 1 & 0 & 1 & 0 & 3 & 2 \\ 0 & 1 & 2 & 3 & 3 & 2 & 1 & 0 & 1 & 0 & 3 & 2 & 2 & 3 & 0 & 1 \end{pmatrix}$$

At least m columns are pairwise disjoint in a covering array with m disjoint columns.

3.2 Block Size Recursive Construction

The construction shown below is known as block-size recursive construction [15] and [13]. This construction uses 2 covering arrays with same alphabet, k and then com-

bining those two CA to form a new covering array with larger parameters. Let $P = (p_{ij})$ be a covering array $CA(n, r, k)$ and $Q = (q_{ij})$ be a covering array $CA(m, s, k)$, now we combine these two covering array using block-size recursive construction to get a covering array $R = CA(m + n, rs, k)$. Let p_i for $i = 1, ..., r$ be the rows of P and let q_j for $j = 1, ..., s$ denote the rows of Q.

$$P = \begin{pmatrix} p_{11} & p_{12} & . & . & . & p_{1n} \\ p_{21} & p_{22} & . & . & . & p_{2n} \\ & & . & & & \\ & & . & & & \\ & & . & & & \\ p_{r,1} & p_{r,2} & . & . & . & p_{r,n} \end{pmatrix}$$

and

$$Q = \begin{pmatrix} q_{11} & q_{12} & . & . & . & q_{1m} \\ q_{21} & q_{22} & . & . & . & q_{2m} \\ & & . & & & \\ & & . & & & \\ & & . & & & \\ q_{s,1} & q_{s,2} & . & . & . & q_{s,m} \end{pmatrix}$$

Now the way this construction of $CA(n + m, rs, k)$ goes about is :

The first s rows of the resultant $CA(n + m, rs, k)$ are row p_1 of P concatenated with rows q_j of Q for $j = 1, ..., s$. The next s rows of the resultant CA are row p_2 of P concatenated with rows q_j of Q for $j = 1, ..., s$. The procedure continues till all the rows of P are covered. The resultant array is shown below :

$$R = \begin{pmatrix}
p_{11} & p_{12} & . & . & . & p_{1n} & q_{11} & q_{12} & . & . & . & q_{1m} \\
p_{11} & p_{12} & . & . & . & p_{1n} & q_{21} & q_{22} & . & . & . & q_{2m} \\
 & & & . & & & & & & . & & \\
 & & & . & & & & & & . & & \\
 & & & . & & & & & & . & & \\
p_{11} & p_{12} & . & . & . & p_{1n} & q_{s,1} & q_{s,2} & . & . & . & q_{s,m} \\
p_{21} & p_{22} & . & . & . & p_{2n} & q_{11} & q_{12} & . & . & . & q_{1m} \\
p_{21} & p_{22} & . & . & . & p_{2n} & q_{21} & q_{22} & . & . & . & q_{2m} \\
 & & & . & & & & & & . & & \\
 & & & . & & & & & & . & & \\
 & & & . & & & & & & . & & \\
p_{21} & p_{22} & . & . & . & p_{2n} & q_{s,1} & q_{s,2} & . & . & . & q_{s,m} \\
 & & & . & & & & & & . & & \\
 & & & . & & & & & & . & & \\
 & & & . & & & & & & . & & \\
p_{r,1} & p_{r,2} & . & . & . & p_{r,n} & q_{11} & q_{12} & . & . & . & q_{1m} \\
p_{r,1} & p_{r,2} & . & . & . & p_{r,n} & q_{21} & q_{22} & . & . & . & q_{2m} \\
 & & & . & & & & & & . & & \\
 & & & . & & & & & & . & & \\
 & & & . & & & & & & . & & \\
p_{r,1} & p_{r,2} & . & . & . & p_{r,n} & q_{s,1} & q_{s,2} & . & . & . & q_{s,m}
\end{pmatrix}$$

To prove that R is a covering array we have the following argument.

Any two distinct rows t_1 and t_2 of the covering array $CA(m+n, rs, k)$ are of the form $a_{i_1}b_{j_1}$ and $a_{i_2}b_{j_2}$. If $i_1 \neq i_2$, then all possible pairs in the $k-$alphabet occur in the first n columns between a_{i_1} and a_{i_2}. If $i_1 = i_2$, then $j_1 \neq j_2$ and all possible pairs in the $k-$alphabet occur in the last m columns between b_{i_1} and b_{i_2}. Hence R is indeed a covering array.

Lemma : For any prime power g, there exists a $CA(2g^2 - g, g(g+1), g)$. Equivalently, for any prime power g and any integer $k \leq$ g(g+1), $CAN(k, g) \leq 2g^2 - g$.

Proof : For k a prime power, use the block-size recursive construction on the following two arrays. First, the $CA(g^2, g+1, g)$ (from the finite field construction) and second, the $CA(g^2, g, g)$ with g disjoint columns. Here, the first g^2 columns cover all the distinct g pairs $(i, i)\ \epsilon\ Z_g \times Z_g$. Therefore, the first g columns of $CA(g^2, g, g)$ need not to be necessarily included in the final covering array. So, removing the mentioned k columns from the $2k^2$ columns gives us the $CA(2g^2 - g, g(g+1), g)$.

Definition : A balanced covering array is a covering array $CA(n, k, g)$ with the property that each alphabet occurs exactly n/g times in each row.

Lemma : For any positive integer i and a prime power g, there exists a balanced covering array $CA(g^2+i(g^2-g), g^i(g+1), g)$ and thus, $CAN(g^i(g+1), g) \leq g^2+i(g^2-g)$.

Proof : We are going to prove this by mathematical induction.
For $i = 1$, $CA(g^2+i(g^2-g), g^i(g+1), g)$ becomes $CA(g^2+g^2-g, g(g+1), g)$ and hence from the above lemma we can say that there exists a $CA(g^2+i(g^2-g), g^i(g+1), g)$ for $i = 1$.
Now, suppose there exists a $CA(g^2+i(g^2-g), g^i(g+1), g)$ for $i = n$ i.e. there exists a covering array

$CA(g^2 + n(g^2 - g), g^n(g + 1), g)$. Now, use the block-size recursive construction on $CA(g^2+n(g^2-g), g^n(g+1), g)$ and $CA(g^2, g, g)$ with g disjoint columns, we can give the similar argument as given in the above lemma that $g^2 + n(g^2 - g)$ columns cover all the distinct g pairs $(i, i)\ \epsilon\ Z_g \times Z_g$ and so the first g columns of $CA(g^2, g, g)$ need not to be necessarily included in the final covering array and hence remove $g^2+n(g^2-g)+1$ from $g^2+n(g^2-g)+g$ to get $CA(g^2+(n+1)(g^2-g), g^{(n+1)}(g+1), g)$. Therefore, it is true for $i = n + 1$ and so this holds for any integer i.

3.3 Group Construction of Covering Arrays

This construction method combines an algebraic construction with a computer search. This construction improves some of the best known upper bounds on the covering array number. The key incentive for this construction is that it uses a small vector to construct a covering array rather than using an entire array.
First, we provide some background results that are used through out this construction.

1. For g a prime power, 2-$CAN(g + 1, g) = g^2$ (This result is given by standard finite field construction for orthogonal arrays).

2. If a 2-$CA(n, k, g)$ exists then there is a 2-$CA(n + g(g - 1), 2k, g)$ by applying the block size recursive construction we get the above result.

3. For g a prime power and $k \leq 2(g + 1)$, 2-$CAN(k, g) \leq 2g^2 - g$.

3.3.1 The Group Construction

This construction appeared in [16] and [1]. This construction involves selecting a subgroup $G < Sym(g)$ and then finding a starter vector $v\ \epsilon\ {Z_g}^k$. $Sym(g)$ is the symmetric group on g elements and G is a subgroup selected. The starter vector v is then used to create a circulant matrix. Then the group G acts on this circulant matrix and produces several matrices which are then concatenated to form a covering array. It is often necessary to add a small matrix C, (generally an array of zeros) to complete the covering array. But the choice of C depends on the group G

and the starter vector v.
The example given below will show how the group construction is done.

Example : Let $G = \{e,(12)\} < Sym(3)$ and $v = (0,2,2,2,1) \epsilon Z_3^5$.

We build the circulant matrix M in the following way taking v as the first column.

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

Now, the two elements of G when act on matrix M, gives

$$M_e = \begin{pmatrix} 0 & 1 & 2 & 2 & 2 \\ 2 & 0 & 1 & 2 & 2 \\ 2 & 2 & 0 & 1 & 2 \\ 2 & 2 & 2 & 0 & 1 \\ 1 & 2 & 2 & 2 & 0 \end{pmatrix} \text{ and } \qquad M_{(12)} = \begin{pmatrix} 0 & 2 & 1 & 1 & 1 \\ 1 & 0 & 2 & 1 & 1 \\ 1 & 1 & 0 & 2 & 1 \\ 1 & 1 & 1 & 0 & 2 \\ 2 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Now M_e and $M_{(12)}$ are concatenated along with the array $C = [0\ 0\ 0\ 0\ 0]^T$. This is done to ensure the coverage of all pairs. Therefore, finally we get $CA(11,5,3)$ as shown below :

$$CA(11,5,3) = \begin{pmatrix} 0 & 0 & 1 & 2 & 2 & 2 & 0 & 2 & 1 & 1 & 1 \\ 0 & 2 & 0 & 1 & 2 & 2 & 1 & 0 & 2 & 1 & 1 \\ 0 & 2 & 2 & 0 & 1 & 2 & 1 & 1 & 0 & 2 & 1 \\ 0 & 2 & 2 & 2 & 0 & 1 & 1 & 1 & 1 & 0 & 2 \\ 0 & 1 & 2 & 2 & 2 & 0 & 2 & 1 & 1 & 1 & 0 \end{pmatrix}$$

To determine which vectors $v = (v_0,v_1,...,v_{k-1})$ can be used as starter vectors to form circulant matrix, consider the sets d_i for $i = 1,2,...,k-1$,

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

where the subscripts are taken modulo k.

Consider the group action of G (taken in the above example) on the pairs of Z_g. The set of orbits $\mathcal{O}$ from this action is shown below :

1. (0,0)

2. {(1,2),(2,1)}
3. {(1,1),(2,2)}
4. {(0,1),(0,2)}
5. {(1,0),(2,0)}

In general, consider the action of the subgroup $G = \langle(1\ 2\ ...g-1)\rangle < Sym(g)$ on the pairs from Z_g, $g+2$ orbits are generated by this group action (shown below):

1. {(0,0)},
2. $\{(0,i) : i = 1,...,g-1\}$,
3. $\{(i,0) : i = 1,...,g-1\}$ and
4. $\{(a,b)$: $a,b \epsilon Z_g \setminus \{0\}$ and $a - b \equiv j(\text{mod } g-1)\}$ for $j = 0,1,...g-2$

for v to be a valid starter vector it should have the property that the set of differences v_j - v_{j+i} for all $(v_j,v_{j+i}) \epsilon d_i$ covers Z_g over $i = 1,...g-1$. In other words, for a starter vector the sets d_i for all $i = 1,...g$ contain at least one element from each of the orbits $\{(a,b)$: $a,b \epsilon Z_g \setminus \{0\}$ and $a - b \equiv j(\text{mod } g-1)\}$ for $j = 0,1,...g-2$. Now, to prove that the matrix produced by the group construction is indeed a covering array, we give the following argument:

Suppose M is a $k \times k$ circulant matrix generated by a starter vector v. Then, for any two rows in M, at least one element from each of the orbits of G (except {(0,0)}) occurs in some column. Suppose $M_g$be the matrix formed by the group action $g \epsilon G$ on M. Now, since any two rows in M have a representative from each orbit of G, for any two rows all pairs from Z_g (except {(0,0)}) will occur in M_g for some $g \epsilon G$. Finally the $k \times 1$ array C with all zero entries is concatenated with M_g for all $g \epsilon G$ to form a covering array.

Chapter 4

Algorithms for Covering Arrays

The main findings in our project are listed in this chapter. As illustrated in the previous chapter, we try to find the starter vector v, but not by using exhaustive search (as exhaustive search is practically impossible for $g \geq 10$) but by heuristic search through which we can get better results for larger g.

The algorithm used was a simple hill-climbing algorithm. It takes an arbitrary vector and counts the number of the orbits of G that occur in all the sets d_i. Then a single entry from v is randomly chosen. This entry is then changed to a random value (from the set Z_g, other than what is already there). If the change increases the number of orbits in the sets d_i, the change is accepted otherwise it is rejected. This procedure is continued until a starter vector is found or if a certain number of changes have been tried.

4.1 Methods

We have done the whole programming in MATLAB and the results obtained are shown in tables in the end of this chapter.
First, by using the hill-climbing algorithm discussed above we found out the starter vectors for many values of k and g . These can also be computed by an exhaustive search (as starter vector is not unique) for values of g = 3,4,...10 as it is feasible. The exhaustive search can only be done upto $g = 10$ and for finding the further starter vectors we have to use the above mentioned hill-climbing algorithm which is fast and easy to use.

To further improve our results substantially, we have written a code in MATLAB, not to construct starter vector but to construct a circulant matrix directly of size $k \times n$ where n is some number which is obviously greater than the number of orbits (this is done keeping in mind that for large k, unnecessary repetition of starter vector lead to larger size of the circulant matrix). Then we can delete some rows (one by one) and keep on checking that it should follow the property that any two rows have representative from each orbit and see how farther we can go in deleting the rows .

4.2 Results Obtained

Our main focus in this thesis is the group construction of covering arrays as discussed earlier.
Using the hill-climbing algorithm as discussed above we have been able to get the starter vectors for many values of k and g. Tables listed in this chapter shows all the values of k and g with $g+1 \leq k \leq 2(g+1)$ for which the starter vector exists that we got and also shows the comparison between existing values and our bounds.

Surprisingly enough, the papers that we followed for referring to this group construction says that for $g = 3$, starter vector only exists for $k = 5,8$ but using our computer search we came across the fact that the starter vectors exist for $k = 5,6,7,8$. Also, for $g = 3$ onwards, it can be checked through our program that the starter vectors do exist for given values of k and the upper bound on covering arrays matches with the given upper bound in the paper.

Using the program that constructs circulant matrix as mentioned in the earlier paragraph, the results match with the already existing bounds that are given in tables below. But, it provides us a good starting point so as to proceed further in our search for a better method that can be used for the construction of optimal covering array.

4.3 Heuristic Search for Starter Vectors

In [2] the starter vectors are found by using exhaustive search but in [3] the method for using hill-climbing algorithm has been introduced, that we have mainly used in our study. Replacing exhaustive search by heuristic search produced good results and in less time mainly for covering arrays with larger g. Through the very simple hill-climbing algorithm, we have managed to find many new upper bounds for covering arrays 2-$CA(n, k, g)$ with $g+ \leq k \leq 2g$. These bounds are shown in Tables at the end of this chapter along with the starter vectors that can be used to produce these bounds. Many of the new bounds are substantially smaller than previously known bounds, that the more relevant part of this study is that this method provides a very efficient tool to build smaller covering arrays with larger alphabet size.

4.4 Table of Bounds on CAN(k,g)

The table below shows the comparison between existing bounds and the bounds that we have obtained using our Matlab programs and computer search.

g =3	Starter Vector	Bound Obtained	Known Bound
k = 4	Starter Vector Do not Exist	–	–
k = 5	0 1 1 2 1	11	13
k = 6	0 1 1 2 1 2	13	13
k = 7	0 1 1 2 1 2 1	15	15
k = 8	0 1 1 2 1 2 1 1	17	15

g = 4	Starter Vector	Bound Obtained	Known Bound
k = 5	0 1 2 2 1	16	22
k = 6	0 1 1 2 1 2	19	24
k = 7	0 2 1 2 1 1 1	22	27
k = 8	0 1 1 2 1 2 1 1	25	27
k = 9	0 1 1 2 1 2 2 2 2	28	29
k = 10	0 1 2 1 1 2 2 2 1 1	31	29

g = 5	Starter Vector	Bound Obtained	Known Bound
k = 7	0 1 2 4 3 1 1	29	41
k = 8	0 1 2 4 3 3 2 3	33	41
k = 9	0 1 2 4 3 3 1 3 3	37	44
k = 10	0 1 2 4 3 3 1 1 2 1	41	45
k = 11	0 1 2 4 3 3 1 1 1 1 3	45	46
k = 12	0 1 2 4 3 3 1 1 1 1 1 3	49	48

g = 6	Starter vector	Bound obtained	Known bound
k = 9	0 2 2 3 2 2 4 1 4	46	48
k = 10	0 1 1 1 1 2 4 3 1 2	51	52
k = 11	0 1 1 1 1 2 4 3 1 2 2	56	64
k = 12	0 1 1 1 1 2 4 3 1 2 2 1	61	67
k = 13	0 1 1 1 1 2 4 3 1 2 2 1 2	66	68
k = 14	0 1 1 1 1 2 4 3 1 2 2 1 2 1	71	68

g = 7	Starter Vector	Bound obtained	known bound
k = 10	0 2 2 2 4 5 2 4 3 1	61	63
k = 11	0 1 1 1 1 2 1 4 6 5 3	67	73
k = 12	0 1 1 1 1 1 2 1 4 6 5 3	73	76
k = 13	0 1 1 1 1 1 1 2 1 4 6 5 3	79	–
k = 14	0 1 1 1 1 1 1 1 2 1 4 6 5 3	85	–
k = 15	0 1 1 1 1 1 1 1 1 2 1 4 6 5 3	91	–
k = 16	0 1 1 1 1 1 1 1 1 1 2 1 4 6 5 3	97	–

g = 8	Starter vector	Bound obtained	known bound
k = 11	0 2 2 3 3 5 3 6 7 4 3	78	80
k = 12	0 2 2 2 3 7 3 7 2 2 7 6	85	99
k = 13	0 1 1 1 1 2 1 3 7 5 1 3 4	92	102
k = 14	0 1 1 1 1 1 2 1 3 7 5 1 3 4	99	104
k = 15	0 1 1 1 1 1 1 2 1 3 7 5 1 3 4	106	–
k = 16	0 1 1 1 1 1 1 1 2 1 3 7 5 1 3 4	113	–
k = 17	0 1 1 1 1 1 1 1 1 2 1 3 7 5 1 3 4	120	–
k = 18	0 1 1 1 1 1 1 1 1 1 2 1 3 7 5 1 3 4	127	–

g =9	Starter vector	Bound obtained	Known bound
k = 13	0 2 2 2 4 3 2 7 3 6 6 4 5	105	120
k = 14	0 2 2 2 2 3 1 6 7 3 2 4 7 8	113	131
k = 15	0 1 1 1 1 1 2 3 2 7 1 5 4 2 5	121	135
k = 16	0 1 1 1 1 1 1 2 3 2 7 1 5 4 2 5	129	145
k = 17	0 1 1 1 1 1 1 1 2 1 2 6 8 5 3 6 7	137	148
k = 18	0 1 1 1 1 1 1 1 1 2 1 2 6 8 5 3 6 7	145	151
k = 19	0 1 1 1 1 1 1 1 1 1 2 1 2 6 8 5 3 6 7	153	–
k = 20	0 1 1 1 1 1 1 1 1 1 1 2 1 2 6 8 5 3 6 7	161	–

g = 10	Starter Vector	Bound obtained	Known bound
k = 15	0 2 2 2 2 5 3 9 2 1 5 6 7 9 5	136	166
k = 16	0 1 1 1 1 1 2 9 5 7 1 5 5 3 2 8	145	177
k = 17	0 2 2 2 2 2 2 3 5 1 6 9 8 3 2 9 6	154	180
k = 18	0 2 2 2 2 2 2 2 3 5 1 6 9 8 3 2 9 6	163	180
k = 19	0 1 1 1 1 1 1 1 1 2 1 4 9 7 8 2 6 8 5	172	180
k = 20	0 1 1 1 1 1 1 1 1 1 2 1 4 9 7 8 2 6 8 5	181	–
k = 21	0 1 1 1 1 1 1 1 1 1 1 1 2 5 6 8 4 8 6 5 2	190	202
k = 22	0 1 1 1 1 1 1 1 1 1 1 1 1 2 5 6 8 4 8 6 5 2	199	202

g = 11	Starter vector	Bound obtained	Known bound
k = 17	0 1 1 1 1 1 3 10 6 9 4 10 9 7 1 4 5	171	221
k = 18	0 2 2 2 2 2 2 2 3 6 9 4 2 4 1 7 6 9 3	181	225
k = 19	0 2 2 2 2 2 2 2 2 3 6 9 4 2 4 1 7 6 9 3	191	231
k = 20	0 2 2 2 2 2 2 2 2 2 3 6 9 4 2 4 1 7 6 9 3	201	231
k = 21	0 2 2 2 2 2 2 2 2 2 2 3 6 9 4 2 4 1 7 6 9 3	211	231
k = 22	0 2 2 2 2 2 2 2 2 2 2 2 3 6 9 4 2 4 1 7 6 9 3	221	231
k = 23	0 2 2 2 2 2 2 2 2 2 2 2 2 3 6 9 4 2 4 1 7 6 9 3	231	231

g = 12	Starter vector	Bound obtained	Known bound
k = 18	0 6 5 10 7 3 9 1 13 2 3 7 6 6 3 1 4 8	199	255
k = 19	0 9 4 4 7 8 11 10 8 11 6 10 6 8 5 3 11 10 4	210	276
k = 20	0 7 9 2 10 10 2 9 8 9 4 9 2 3 1 4 5 6 1 9	221	276
k = 21	0 6 2 10 2 5 3 8 10 5 8 7 6 6 3 4 8 8 3 8 10	232	276
k = 22	0 1 1 3 10 6 9 4 10 9 1 1 1 4 8 6 3 2 7 8 4 11	243	288
k = 23	0 1 1 1 3 10 6 9 4 10 9 1 1 1 1 9 2 11 5 11 1 6 5	254	288
k = 24	0 2 2 2 2 4 11 7 10 5 11 10 2 2 2 2 7 6 3 1 2 6 7 3	265	288
k = 25	0 8 11 8 9 3 3 2 2 11 2 6 6 6 2 8 11 2 7 2 7 5 10 10 11	276	288

Similarly, we can find many starter vectors for higher values of g by this method and it can be seen clearly in the tables, this method of construction of covering array improves the upper bound on the covering array number significantly.

4.5 Conclusions and Further Suggesstions

The upper bounds found here need not be the best possible, but that certainly brings us closer to the optimal. The covering arrays found here can also be used as good starting points to may be able to produce much tighter bounds. The use of a variety of algorithms for heuristic search can help us to find starter vectors which can in turn, find good upper bounds and in lesser time.
Another area that can be explored is the group action, is that the group, Z_g that we have used is not transitive but it produced good results quickly. Groups like $A_3 < S_3$ can be tried for results in the case of strength 3 covering arrays. Eventually, if a method can be developed that will take a group action, find the orbits and then search for useful starter vectors for strength 3 covering arrays, that can work as good starting point to explore for success in strength 3.

References

[1] K. Meagher, Group construction of covering arrays-part 2, *University of Ottawa, site Technical report.*

[2] C. A. tables, *www.public.asu.edu/ ccolbou/src/tabby/catable.html.*

[3] N.Leveson, C.S.Turner, An investigation of the therac-25 accidents, *IEEE Computer* 26 (1993) 18–41.

[4] L. J.L., Ariane 5 Flight 501 Failure , Report by inquiry board.

[5] E. A., The mathematics of the pentium division bug (1997) 54–67.

[6] N. S. A.S.Hedayat, J.Stufken, Orthogonal arrays, *Springer Series in statistics* Springer-Verlag (1999) 237–329.

[7] S.Kageyama, On orthogonal arrays attaining rao's bound, *Journal of Statistics planning and Inference* 19 (1988) 395–400.

[8] K. Meagher, Covering arrays on graphs: Qualitative independence graphs and extremal set partition theory, *PhD Thesis*, University of Ottawa (2005) Canada.

[9] M.Chateauneuf, D. Kreher, On the state of strength three covering arrays, *Journal of Combinatorial Designs* 10 (7) (2002) 217–238.

[10] M. D.M.Cohen, S.R.Dalal, G.C.Patton, The aetg system : An approach to testing based on combinatorial design, *IEEE Transactions on software engineering* 23 (1997) 437–444.

[11] A. Hartman, Software and hardware testing using combinatorial covering suites, *Graph theory, Combinatorics and Algorithms : Interdisciplinary Applications* 34 (2005) 237–266.

[12] D.Stinson, Combinatorial designs: Constructions and analysis.

[13] B.Stevens, E.Mendelsohn, New recursive methods for transversl covers, Journal of Combinatorial Designs 7 (1999) 185–203.

[14] C.J.Colbourn, Combinatorial aspects of covering arrays, Le Mathematiche (Catania) 23 (2004) 437–444.

[15] A. P. S.Poljak, V.Rodl, On qualitaively independent partitions and related problems, Discrete Applied mathematics. 6 (1983) 193–205.

[16] K. Meagher, B. Stevens, Group construction of covering array, *Journal of combinatorial designs* 13(1) (2004) 70–77.

Appendix A

Important MATLAB Programmes

Here we have shown the important Matlab Programmes that are used in generating the result, and are crucial for this project. Rest of the work done in MATLAB is not been shown here as they are not very important in generating the results that are shown in the table given in Appendix A.

A.1 main code(count2m.m)

clc clears the window
clear all clears the memory of all the variables
k=input('input length of vector'); input length of vector from user in k
g1=input('number of symbols'); input number of symbol from user in g1
g=g1-1; g=g1-1
m=zeros(1,k); declare a zeros array named m with 1 row and k column
n=zeros(1,k); declare a zeros array named n with 1 row and k column
for i=1:k-1 command for loop to run i from 1 to k-1
m(1,i+1)=input('input any integer among symbols except 0 ');
input from user to fill m from 2nd to last column
end loop end after running all the iterations
disp('initial vector'); displays 'initial vector' on the screen window of user
n=m; copies m to n
disp('initial count'); displays 'initial count' on the screen window of user
s0=r1(n,g1);
calls function named r1 by passing parameters n and g1 to r1
and stores the return value by function r1 in s0.
if sum(s0)<(g1+1)*(k-1) ; Checks for the condition, where sum(s0) equals the sum of components of s0
disp('invalid starter vector'); If above conditions satisfied it displays 'invalid starter vecter'
end end of if condition
if sum(s0)==(g1+1)*(k-1); Again checks for the condition
disp('valid starter vector'); If above condition satisfied it dis-

```
plays 'valid starter vector'
end                                                    end of if condition
n1=zeros(1,k*k);              declare a zeros array named n1 with 1 row and k*k
coloumn
n1(1)=n(1);                          copies column 1 value of n to position 1 of n1
for i=1:k*k-1                          command for loop to run i from 1 to k*k-1
n1(i+1)=n(mod(i,k)+1);          copies coloumn (mod(i,k)+1) value to position i+1
of n1
end                                          loop end after running all the iterations
ca=zeros(k,k);                        declare a zeros array named ca with k row
and k coloumn
for i=1:k                                    command for loop to run i from 1 to k
for j=1:k                                    command for loop to run j from 1 to k
ca(i,j)=n1(1,i+j-1);          copies coloumn (i+j-1) value of n1 to position (i,j) of ca
end                                                          loop for j end
end                                                          loop for i end
ca                                                        displays contents of ca
c1=zeros(1,(k*(k-1)/2),k);          declare a 3-d zeros array named c1 with 1 row and
(k*(k-1)/2) coloumn and k stacks
for l=1:k                                          runs loop for l from 1 to k
A=zeros(2,k,(k*(k-1)/2));           declare a 3-d zeros array named A with 2 row and
k coloumn and (k*(k-1)/2) stacks
Z=1;                                                          z=1
for i=1:k-1                                      runs loop for i from 1 to k-1
for j=i+1:k                                      runs loop for j from i+1 to k
A(1,:,Z)=ca(i,:);                  copies ca i^th row to A 1st row and z stack
A(2,:,Z)=ca(j,:);                  copies ca j^th row to A 2nd row and z stack
Z=Z+1;                                                  z transforms to z+1
end                                                              j loop end
end                                                              i loop end
A(:,l,:)=[];                                  A transforms with deleted l coloumn
o=orbit2(g1);          calls for function named orbit2 and store the return value in o
c0=count2(g1,k-1,(k*(k-1)/2)+1,A,o);          calls for function named count2 and
store the return value in c0
c1(1,:,l)=c0(1,:);                  copies c0 row 1 to c1 row 1 and stack l
end                                                              loop l ends
```

A.2 function(r1.m)

```
function [s0]=r1(n,g1)
f=n;
k=length(n);
v=zeros(1,2*k);
for i=1:k
v(1,i)=f(1,i);
```

```
v(1,i+k)=f(1,i);
end
o=zeros(2,g1-1,g1+1);
o=orbit2(g1);
d=zeros(2,k,k-1);
for i=1:k
d(1,i,:)=v(1,i);
end
for j=1:k-1
for i=1:k
d(2,i,j)=v(1,i+j);
end
end
s0=count2(g1,k,k,d,o);
```

A.3 function(count2.m)

```
function [ic]=count2(g1,k,k1,d,o)
count=zeros(1,g1+1,k1-1);
for i=1:k1-1
for j=1:k

for q=1:g1+1
for r=1:g1-1
if(d(:,j,i)==o(:,r,q))
count(1,q,i)=1;
end
end
end
end
end
t=zeros(1,k1-1);
for i=1:k1-1
for m=1:g1+1
t(1,i)=t(1,i)+count(1,m,i);
end
end
ic=t;
```

A.4 function(orbit2)

```
function [z1]=orbit2(g1)
g=g1-1;
v=zeros(1,2*g);
```

```
for i=1:g
v(i)=i;
v(g+i)=i;
end
z1=zeros(2,g,g+2);
for k=1:g
for i=1:g
z1(1,i,k)=v(i);
z1(2,i,k)=v(i+k);
end
end
for i=1:g
z1(2,i,g+1)=v(i);
z1(1,i,g+2)=v(i);
end
```