Covering Arrays and Generalizations

A thesis

Submitted in partial fulfillment of the requirements

Of the degree of

Doctor of Philosophy

By

Yasmeen Akhtar

20113108

INDIAN INSTITUTE OF SCIENCE EDUCATION AND RESEARCH PUNE

To my beloved mother

And

To the loving memory of my father
For their love, endless support, and encouragement

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

[illegible]

Yasmeen Akhtar

20113108

Date: July 22, 2016

Certificate

Certified that the work incorporated in the thesis entitled “Covering Arrays and Generalizations” submitted by Yasmeen Akhtar was carried out by the candidate, under my supervision. The work presented here or any part of it has not been included in any other thesis submitted previously for the award of any degree or diploma from any other University or Institution.

Soumen Maity

Dr. Soumen Maity

Date: July 22, 2016

Acknowledgements

It is a great pleasure to express my deep sense of gratitude to Dr. Soumen Maity for supervising my research work, for his valuable suggestions and the constant encouragement I received from him.

I am grateful to Prof. Charles Colbourn, Arizona State University, and Reshma Chandrasekharan, Indian Institute of Science Education and Research, Pune, for their kind permission to include the results of the joint papers in this thesis; for going through parts of my thesis and suggesting valuable improvements.

I am also grateful to Prof. Jaikumar Radhakrishnan, Tata Institute of Fundamental Research, Mumbai and Dr. Vivek Mallick, Indian Institute of Science Education and Research, Pune for being in my research advisory committee and for their valuable suggestions. I would like to thank Prof. A. Raghuram for providing me with requisite facilities to carry out my doctoral research during his tenure as the chair of the department. Thanks are also due to Dr. Lucia Moura and Dr. Sebastian Raaphorst, University of Ottawa for suggesting useful improvements.

I am grateful to the Indian Institute of Science Education and Research, Pune, for providing ample facilities for my research. I gratefully acknowledge financial support from the Council of Scientific and Industrial Research (CSIR), India, during the work under CSIR Junior and senior research fellow scheme. Thanks are also due to National PARAM Supercomputing Facility, Center for Development of Advance Computing (C-DAC) for providing me supercomputing facility.

Yasmeen Akhtar

Pune, India, 2016

Abstract

Covering arrays have been successfully applied in the design of test suites for testing systems such as software, circuits, and networks, where failures can be caused by the interaction between their parameters. There has been a great deal of research on covering arrays for last thirty years. Much research has been carried out in developing effective methods to construct covering arrays and generalizations of covering arrays. 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 g symbols such that every $t \times n$ subarray contains every $t \times 1$ column on g symbols at least once. It is desirable in most applications to minimize the size n. A covering array is *optimal* if it has the minimum number of columns among all covering arrays with the same degree, strength, and order. In this dissertation, we give an algebraic construction that can be used to build strength four covering arrays. The construction given here yields many new upper bounds on the size of optimal covering arrays when $g = 3$.

For software or hardware testing applications, each row of a covering array corresponds to a parameter; each column corresponds to a test case, and the g symbols correspond to the values for each parameter. In most software development environments, we have limited time, computing, and human resources to perform the testing of a system. To model this situation, we consider the problem of creating a best possible testing array (covering the maximum number of t-way parameter-value configurations) within a fixed number of test cases. If the testing array is a covering array, then configuration coverage is 100%. We present algebraic constructions for testing arrays with high 3- and 4-way configuration coverage.

Two vectors $u, v \in \mathbb{Z}_g^n$ are *qualitatively independent* if for each ordered pair $(a,b) \in \mathbb{Z}_g \times \mathbb{Z}_g$ there is a position $1 \leq i \leq n$ such that $\big(u(i), v(i)\big) = (a,b)$. A strength two covering array is an array with the property that every pair of rows are qualitatively independent. A *covering array on a graph* is an array with a row for each vertex of the graph with the property that any two rows which correspond to adjacent vertices are qualitatively independent. Given a graph G and a positive integer g, a covering array on G with minimum size n is called *optimal*. Our primary focus is with constructions that make optimal covering arrays on large graphs that are obtained from a product of smaller graphs. We consider four most extensively studied graph products in the literature and give upper and lower bounds on the size of an optimal covering array on a product graph. We find families of graphs for which the size of a covering array on a product graph achieves the lower bound with respect to the Cartesian product. In addition, we present a polynomial time approximation algorithm for constructing covering arrays on graphs having k prime factors with respect to the Cartesian product.

We consider a generalization of covering arrays on graphs to higher strength, called mixed covering arrays on 3-uniform hypergraphs. The addition of a graph or hypergraph structure to covering arrays makes it possible to use methods from graph and hypergraph theory to study covering arrays. We introduce four hypergraph operations that allow us to add new vertices to a hypergraph while preserving the size of a mixed covering array on the hypergraph. Using these operations, for the case in which H is a 3-uniform α-acyclic hypergraph, a 3-uniform interval hypergraph, a 3-uniform conformal hypertree having a binary tree as host tree, a 2-tree hypergraph, or a 3-uniform loose cycle, we construct an optimal mixed covering array on H.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Covering arrays have been the focus of much research for last thirty years. They are natural generalizations of the well-known and well-studied orthogonal arrays [49]. Covering arrays are computationally difficult to find and have been studied for their applications to software testing, hardware testing, drug screening, and in areas where interactions of multiple parameters are to be tested [41, 47, 48, 52, 53, 58, 70]. To begin, we give a definition of covering arrays. 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 g symbols such that every $t \times n$ subarray contains each t-tuple from the set of g symbols at least once as a column. A covering array is *optimal* if it has the smallest possible number n of columns. This number is the *covering array number*,

$$t\text{-}CAN(k,g) = \min\Big\{n \ : \ \text{there exist a } t\text{-}CA(n,k,g)\Big\}.$$

It is desirable in most applications to minimize the size n of covering arrays. For example, the following array is a covering array for $t = 2$, $g = 2$ and $k = 4$, because whichever two rows out of the four rows are chosen, all possible pairs $00, 01, 10$ and 11 come up at least once:

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

The above covering array with five columns is, in fact, optimal, as set forth by a paper by Kleitman and Spencer [55]. For the special case of strength two binary covering arrays (covering arrays with $t = 2$ and $g = 2$), the exact sizes of the smallest arrays are known [54, 55, 80]. Except in this special case for $t = 2$ and $g = 2$, it is generally unknown what are the sizes of optimal covering arrays. A database maintained by Charles Colbourn at Arizona State University lists the best-known sizes of covering arrays for a broad range of configurations ranging from $t = 2$ to $t = 6$ [31]. Much of the work on covering arrays has been done on developing constructions for them. These constructions include the use of finite field theory [17, 94], group theory [20, 21, 60, 69], combinatorial recursive techniques [83, 104], coding theory [91], extremal set theory [66] and heuristic search algorithms [14, 15, 26, 74]. In this dissertation, we consider the following four combinatorial problems related to covering arrays: An algebraic construction of strength four covering arrays; Testing arrays with high coverage measure; Covering arrays on product graphs; Mixed covering arrays on 3-uniform hypergraphs. In Section 1.4, we describe the problems considered and a brief outline of the solutions. Covering arrays are closely related to other designs like Latin squares, orthogonal arrays, and transversal designs. In the following section, we give an overview of covering arrays and related designs.

1.1 Covering arrays and some related designs

Covering arrays are a relaxation of the well-known and well-studied orthogonal arrays. In 1949, C.R. Rao [78] introduced orthogonal arrays for designing statistical experiments. Orthogonal arrays have great significance in the field of design of experiments; in an experiment based on orthogonal array the estimated effect of any factor is statistically independent of the estimated effect of any other factor [49]. Orthogonal arrays are closely related to Latin squares and transversal designs. The results in Section 1.1.1 can be found in [12].

1.1.1 Latin squares

A *Latin square* of order n is identified as an $n \times n$ square, the n^2 cells of which are occupied by n distinct symbols such that each symbol occurs once in each row and once in each column. For example, the square in Figure 1.1 is a Latin square of order 6, the symbols for the square being $0,1,2,3,4,5$. Two Latin squares of order *n*, one with symbols A, B,

1	3	0	4	2	5
3	0	2	1	5	4
4	5	3	0	1	2
0	2	4	5	3	1
2	1	5	3	4	0
5	4	1	2	0	3

Figure 1.1: A Latin square of order 6

C, ..., and one with symbols a, b, c, ..., are *orthogonal* if superimposing them leads to a square array containing all n^2 possible pairs (A,a), (A,b), ..., (B,a), (B,b), ... Figure 1.2 shows two orthogonal Latin squares of order 5. In 1779 Euler could not find a pair of

A	B	C	D	E
B	C	D	E	A
C	D	E	A	B
D	E	A	B	C
E	A	B	C	D

a	b	c	d	e
c	d	e	a	b
e	a	b	c	d
b	c	d	e	a
d	e	a	b	c

Figure 1.2: Two orthogonal Latin squares of order 5

orthogonal Latin squares of order 6 and he conjectured that there is no pair of orthogonal Latin squares of order 6, and in fact there is no pair of any order that is twice an odd number.

The Euler Conjecture: For $n \equiv 2 \pmod 4$, there is no pair of orthogonal Latin squares of order *n*.

In 1900 Tarry managed to list all Latin squares of order 6 and showed that Euler was correct that there is no pair of orthogonal Latin squares of order 6. The general form of Euler's conjecture remained unsolved till 1960 when Bose, Shrikhande, and Parker [13] showed that the rest of Euler's conjecture was wrong. In fact, orthogonal Latin squares exist for orders 10,14,18, and so on. Thus orthogonal Latin squares exist for every order n except $n = 1,2$, and 6.

A set of Latin squares (all of the same order), any two of which are orthogonal, is said to be a *set of mutually orthogonal Latin squares*. It has been proved that there cannot exist a set of more than $n-1$ mutually orthogonal Latin squares of order n. A set of $n-1$ mutually orthogonal Latin squares of order n is said to be a *complete set of mutually orthogonal Latin squares*. When n is a prime power, we have the following theorem.

Theorem 1.1.1. [97] *For any prime power n there exists a complete set of mutually orthogonal Latin squares of order n.*

Example 1.1.1. Let $n = 3$; then the two mutually orthogonal Latin squares of order 3 are exhibited in Figure 1.3.

$$\boxed{\begin{array}{ccc} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{array}} \qquad \boxed{\begin{array}{ccc} 0 & 1 & 2 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{array}}$$

Figure 1.3: Complete set of mutually orthogonal Latin squares of order 3

Let $N(n)$ be the maximum number of mutually orthogonal Latin squares of order n. Chowla, Erdos and Straus [25] generalized the method used in [13] to show that $\lim_{n\to\infty} N(n) = \infty$. Currently, it is known that $N(2) = 1$, $N(6) = 1$ and $N(10) \geq 2$, and for all other values of n, $N(n) \geq 3$ [97]. For more information see [33, 88].

1.1.2 Orthogonal arrays

Orthogonal arrays are combinatorial structures that have been used in the statistical design of experiments for over 60 years. The results in this section can be found in [37, 49, 89].

Definition 1.1.1. An *orthogonal array* of *size n*, with *k factors*, *g symbols*, *strength t*, and *index* λ, denoted $OA(n,k,g,t)$, is a $k \times n$ array with entries from $\mathbb{Z}_g = \{0,1,\ldots,g-1\}$ with the property that in every $t \times n$ subarray, each t-tuple from $\mathbb{Z}_g$ appears precisely $\lambda = \frac{n}{g^t}$ times. An $OA(n,k,g,t)$ is also denoted by $OA_\lambda(k,g,t)$. If t is omitted, it is understood to be 2. If λ is omitted, it is understood to be 1.

It is clear that an orthogonal array of index 1 is a special case of covering array, since in a covering array each t-tuples from the set of g symbols is required to appear at least once. Thus, an orthogonal array is always an optimal covering array. Orthogonal arrays of strength 2 and index 1 have been well studied as they are equivalent to mutually orthogonal Latin squares of order g.

It is not difficult to construct an $OA(k+2,g)$ from k mutually orthogonal Latin squares of order g and vice versa [17]. Suppose these k mutually orthogonal Latin squares of order g are named $L_1,L_2,\ldots,L_k$, defined on symbols $\{0,1,\ldots,g-1\}$, and having rows and columns labelled $\{0,1,\ldots,g-1\}$. For every $i,j \in \{0,1,\ldots,g-1\}$, construct a $(k+2)$-tuple

$$\big(i,j,L_1(i,j),L_2(i,j),\ldots,L_k(i,j)\big).$$

Then form an array A whose columns consists of these g^2 $(k+2)$-tuples. It is easy to verify that A is an $OA(k+2,g)$.

Example 1.1.2. Superimposing $k = 2$ mutually orthogonal Latin squares of order 3 we obtain:

0, 0	1, 1	2, 2
1, 2	**2, 0**	0, 1
2, 1	0, 2	1, 0

The highlighted entry corresponds to 4-tuple $(1,1,2,0)$: row label 1, column label 1, entry $(2,0)$. Continuing in this way we obtain $OA(4,3)$ as shown below:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 2 & 2 & 2 \\ 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 \\ 0 & 1 & 2 & 1 & 2 & 0 & 2 & 0 & 1 \\ 0 & 1 & 2 & 2 & 0 & 1 & 1 & 2 & 0 \end{bmatrix}$$

It is well-known that there exists a set of $g-1$ mutually orthogonal Latin squares of order g if and only if there exists a finite projective plane of order g. It is also well-known that a finite projective plane exists when the order g is a power of a prime, that is $g = p^m$ for $m \geq 1$. The construction of projective planes of prime order was generalized by Bush [17] who proved the existence of orthogonal array $OA(g^t, g+1, g, t)$ when g is a prime power. We give a proof of this in Section 2.2.1.

1.1.3 Transversal designs

The orthogonal arrays and transversal designs are equivalent objects. The results in this section can be found in [96].

Definition 1.1.2. Let $k \geq 2$ and $g \geq 1$ be integers. A *transversal design* $TD(k,g)$ is a triple $(\mathcal{X}, \mathcal{G}, \mathcal{B})$ such that the following properties are satisfied:

1. $\mathcal{X}$ is a set of kg elements called *varieties*.
2. $\mathcal{G}$ is a partition of $\mathcal{X}$ into k subsets of size g each. That is, $\mathcal{G} = \{G_1, G_2, \ldots, G_k\}$. The sets G_i are called *groups*.
3. $\mathcal{B}$ is a set of k-subsets of $\mathcal{X}$ called *blocks*. Each block intersects each group in exactly one variety.
4. Every pair of varieties from distinct groups is contained in exactly one block.

Example 1.1.3. Let $k = 4$ and $g = 3$. Let

$$\mathcal{X} = \{(0,0),(0,1),(0,2),(0,3),(1,0),(1,1),(1,2),(1,3),(2,0),(2,1),(2,2),(2,3)\}$$

be partitioned into groups G_0, G_1, G_2, G_3 where

$$G_0 = \{(0,0),(1,0),(2,0)\} \qquad G_1 = \{(0,1),(1,1),(2,1)\}$$
$$G_2 = \{(0,2),(1,2),(2,2)\} \qquad G_3 = \{(0,3),(1,3),(2,3)\}.$$

Then the following blocks form a transversal design $TD(4,3)$:

$$B_0 = \{(0,0),(0,1),(0,2),(0,3)\} \qquad B_1 = \{(0,0),(1,1),(1,2),(1,3)\}$$
$$B_2 = \{(0,0),(2,1),(2,2),(2,3)\} \qquad B_3 = \{(1,0),(0,1),(1,2),(2,3)\}$$
$$B_4 = \{(1,0),(1,1),(2,2),(0,3)\} \qquad B_5 = \{(1,0),(2,1),(0,2),(1,3)\}$$
$$B_6 = \{(2,0),(0,1),(2,2),(1,3)\} \qquad B_7 = \{(2,0),(1,1),(0,2),(2,3)\}$$
$$B_8 = \{(2,0),(2,1),(1,2),(0,3)\}.$$

An orthogonal array $OA(k,g)$ is equivalent to a transversal design $TD(k,g)$. We show how to construct a $TD(k,g)$ from an $OA(k,g)$. Let A be an orthogonal array $OA(k,g)$ with symbols from $\mathbb{Z}_g$. Label the rows of A as $0,1,\ldots,k-1$ and columns of A as $0,1,\ldots,g^2-1$. Define

$$\mathcal{X} = \{0,1,\ldots,g-1\} \times \{0,1,\ldots,k-1\}.$$

For $0 \le i \le k-1$, define $G_i = \{0,1,\ldots,g-1\} \times \{i\}$, and then

$$\mathcal{G} = \{G_i \ : \ 0 \le i \le k-1\}.$$

For $0 \le j \le g^2-1$, define $B_j = \{(A(i,j),i) \ : \ 0 \le i \le k-1\}$, and then

$$\mathcal{B} = \{B_j \ : \ 0 \le j \le g^2-1\}.$$

It is easy to prove that $(\mathcal{X},\mathcal{G},\mathcal{B})$ is a $TD(k,g)$. The construction can be reversed to obtain an orthogonal array from a transversal design [33, 97]. The above-mentioned transversal design $TD(4,3)$ is derived from the orthogonal array $OA(4,3)$ given in Example 1.1.2. The equivalence between above mentioned three designs namely Latin squares, orthogonal arrays and transversal designs can be stated as follows.

Theorem 1.1.2. [33, 96] *Let $k \geq 3$ and $g \geq 2$ be two positive integers. Then the existence of any one of the following designs implies the existence of the other two designs:*

1. *k mutually orthogonal Latin squares of order g,*

2. *an $OA(k+2,g)$,*

3. *a $TD(k+2,g)$.*

One well-known generalization of transversal design is called *transversal cover*; this generalization is similar to the relaxation of orthogonal arrays to covering arrays. A transversal cover $TC(k,g)$ is a triple $(\mathcal{X},\mathcal{G},\mathcal{B})$ with the same properties as that of a transversal design $TD(k,g)$ except the property that any pair of varieties from distinct groups occurs in *exactly* one block is relaxed to any pair of varieties from distinct groups occurs in *at least* one block. Note that the number of blocks in a transversal cover $TC(k,g)$ is at least g^2. Thus, a transversal cover $TC(k,g)$ that has exactly g^2 blocks is a transversal design $TD(k,g)$. Stevens [93], Stevens, Moura and Mendelsohn [95] gave a detailed study of these designs and bounds on the fewest blocks possible in a transversal cover. Here we give an example of a transversal cover.

Example 1.1.4. [93] Let $\mathcal{X} = \mathbb{Z}_8$ be the set of varieties. Let $\mathcal{G}$ be a partition of $\mathcal{X}$ into 4 groups of size 2 namely,

$$G_0 = \{0,1\} \qquad G_1 = \{2,3\} \qquad G_2 = \{4,5\} \qquad G_3 = \{6,7\}$$

and $\mathcal{B}$ be the set of 5 blocks of size 4 as given below:

$$B_0 = \{0,2,4,6\} \qquad B_1 = \{1,3,5,6\} \qquad B_2 = \{1,3,4,7\}$$
$$B_3 = \{1,2,5,7\} \qquad B_4 = \{0,3,5,7\}.$$

Then $(\mathcal{X},\mathcal{G},\mathcal{B})$ is a transversal cover $TC(4,2)$.

1.1.4 Covering arrays

Orthogonal arrays exist only for specific combination of parameters n,k,g,t and $\lambda = \frac{n}{g^t}$. In order to widen the use of orthogonal arrays to a larger range of problems, the balance requirement that every g^t tuple to appear precisely $\lambda = \frac{n}{g^t}$ times was relaxed to the requirement that every g^t tuple to appear at least $\lambda = \frac{n}{g^t}$ times; and the resultant structure named a covering array. In practice, we are most interested in cases where $\lambda = 1$.

Definition 1.1.3. 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 g symbols such that every $t \times n$ subarray contains each possible t-tuple from the set of g symbols at least once as a column.

Figure 1.4 shows an example of a covering array of strength three with degree 4 and order 2. We can see that every three rows of this array contain all eight possible 3-tuples $(000, 001, 010, 011, 100, 101, 110, 111)$ at least once.

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

Figure 1.4: 3-$CA(8,4,2)$

The number of columns n in a t-$CA(n,k,g)$ is called the *size* of the covering array. It is desirable in most applications to use a covering array with the smallest size. The smallest possible size of a covering array for fixed parameters t,k and g is denoted as

$$t\text{-}CAN(k,g) = \min\Big\{n \ : \ \exists\ t\text{-}CA(n,k,g)\Big\}.$$

A covering array t-$CA(n,k,g)$ with $n = t$-$CAN(k,g)$ is said to be *optimal*. For a covering array t-$CA(n,k,g)$ it is trivial that $n \geq g^t$, and thus

$$t\text{-}CAN(k,g) \geq g^t.$$

This lower bound is of little use as it does not tell us how fast t-$CAN(k,g)$ grows as a function of k. The only case where tight lower bounds have been obtained is when $g = t = 2$. This result has been discovered by Rényi [80] when n is even, and independently by Katona [54], and Kleitman and Spencer [55] for all n. For $k > 1$, we have

$$2\text{-}CAN(k,2) = \min\left\{n \;:\; \binom{n-1}{\lfloor \frac{n}{2} - 1 \rfloor} \geq k\right\}.$$

This lower bound is obtained using Sperner's lemma [92] when n is even and using Erdos-Ko-Rado theorem [40] when n is odd. Stevens [93], Stevens, Moura, and Mendelsohn [95] have also proved some other lower bounds on the size of covering arrays. Stevens, Moura, and Mendelsohn [95] established that

$$2\text{-}CAN(k,g) \geq g^2 + 3$$

when $3 \leq g \leq k-3$. Many of their results provide useful information on small parameter sets.

For strength three, Kleitman and Spencer [55] and Sloane [91] used results from binary intersecting codes and probability theory to obtain the bounds on the size of binary covering arrays. It says that for $t = 3$ and large k, the minimum n satisfies

$$3.21256 < \tfrac{n}{\log k} < 7.56444.$$

However, for fixed strength t and order g, probabilistic methods establish the following result [44, 54, 55]:

$$t\text{-}CAN(k,g) = \Theta(\log k).$$

Research has been done to determine the value of

$$d(t,g) = \limsup_{k \to \infty} \frac{t\text{-}CAN(k,g)}{\log_2 k}.$$

The exact value of $d(t,g)$ is only known when $t = 2$ and $g \geq 2$ [43], which is equal to $\frac{g}{2}$. In general for any strength $t \geq 2$ and a positive integer g, an upper bound on $d(t,g)$ is given by Godbole *et al.* [44] as follows:

$$d(t,g) \leq \frac{t-1}{\log_2 \frac{g^t}{g^t-1}}.$$

Much of the research on covering arrays involves developing new constructions and bounds on the size of covering arrays [17, 20, 21, 26, 27, 60, 64, 69, 72, 73, 74, 83, 91, 94, 104]. Many algorithms and constructions have been developed for computing covering arrays, but there is no uniformly best method, in the sense of always computing the smallest possible covering array. For a good up to date survey on covering arrays see [47]. Improving the best-known sizes of covering arrays for fixed strength, degree and order is considered to be a primary concern in the study of covering arrays.

Just as strength two orthogonal arrays are equivalent to transversal designs, strength two covering arrays are equivalent to transversal covers. It is not difficult to construct a 2-$CA(n,k,g)$ from a $TC(k,g)$ where n is the number of blocks in the transversal cover. A covering array is formed by associating the same g-ary alphabets $\{0,1,\ldots,g-1\}$ with elements of each group and then listing the blocks explicitly as the columns of the array. We give an example to explain the method.

Example 1.1.5. [93] Below is a binary covering array 2-$CA(5,4,2)$ derived from the transversal cover $TC(4,2)$ given in Example 1.1.4. Here varieties $0,2,4,6$ are associated with symbol 0 and varieties $1,3,5,7$ are associated with symbol 1, and block B_i corresponds to the ith column in the covering array.

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

1.2 Generalizations of covering arrays

In this section we consider some generalizations of the problem of constructing covering arrays with smaller size. Covering arrays have applications in the design of test suites for testing systems such as software, circuits, and networks, where failure can be caused by the interaction between their parameters. These generalizations are motivated by their applications in software and hardware testing. We look at the rows of a covering array in a different way which we also use throughout this thesis.

Definition: Let $g_1 \geq g_2 \geq \ldots \geq g_t$ be positive integers. A set of t vectors $\{x_1, \ldots, x_t\}$ where $x_i \in \mathbb{Z}_{g_i}^n$, $1 \leq i \leq t$, is said to be *t-qualitatively independent* if for every t-tuple $(a_1, \ldots, a_t) \in \mathbb{Z}_{g_1} \times \cdots \times \mathbb{Z}_{g_t}$, there exists $1 \leq j \leq n$ such that $(x_1(j), \ldots, x_t(j)) = (a_1, \ldots, a_t)$.

1.2.1 Mixed covering arrays

Mixed covering arrays are a generalization of covering arrays that allow different values in different rows. This generalization is typically based on the most practical constraint in a testing process where different parameters in a system take a different number of values.

Definition 1.2.1. Let $n, k, g_1, \ldots, g_k$ be positive integers. A *mixed covering array* of strength t, denoted by t-$CA(n, k, \prod_{i=1}^{k} g_i)$, is a $k \times n$ array C with entries from $\mathbb{Z}_{g_i}$ in row i, such that any t distinct rows of C are t-qualitatively independent.

The parameter n is called the size of the array. The main problem is to construct mixed covering arrays with minimum size n for the given values of k, t and g_i's. An obvious lower bound for the size of a mixed covering array is $\prod_{i=1}^{t} g_i$ where $g_1, \ldots, g_t$ are the largest t values, in order to guarantee that the corresponding t rows be t-qualitatively independent. Moura *et al.* [72] developed a theory for mixed covering arrays and presented a detailed study of constructing optimal mixed covering arrays with specific parameters for $t = 2$ and $k \leq 5$. This problem is further discussed by Colbourn *et al.* in [34]. The case $t = 3$ is studied by Colbourn *et al.* in [35] for $k \leq 6$.

1.2.2 Covering arrays on graphs

Another generalization of covering arrays are covering arrays on graphs. This is useful in testing applications where two specific parameters do not interact. Then it is not necessary that each possible parameter-value configuration for these two parameters be tested, which allows reductions in the number of required test cases. We can use a graph structure to describe which pairs of parameters need to be tested. A *covering array $CA(n,G,g)$ on a graph G* with alphabet size g is a $|V(G)| \times n$ array over $\mathbb{Z}_g$. Each row in the array corresponds to a vertex in the graph G. The array has the property that any two rows which correspond to adjacent vertices in G are qualitatively independent. The size of the smallest possible covering array on a graph G is called as *g-qualitative independence number of G or g-ary covering array number of G*, given by

$$CAN(G,g) = \min_{n\in\mathbb{N}} \big\{ n \ : \ \text{there exists a } CA(n,G,g) \big\}.$$

A $CA(n,G,g)$ of size $n = CAN(G,g)$ is called optimal. A quick observation implies that a covering array on a complete graph is a covering array. Here also a classical problem is to construct covering arrays on graphs with the minimum number of columns n for a given graph G and an integer g. Serroussi and Bshouty [90] showed that finding an optimal covering array on a graph is NP-hard even for the binary case. Stevens gave some basic results on covering arrays on graphs in his Ph.D thesis [93]. Meagher and Stevens [68] developed further research in this topic and presented several bounds for the g-qualitative independence number of G.

1.2.3 Mixed covering arrays on graphs

Mixed covering arrays on graphs are structures that generalize the notion of mixed covering arrays as well as covering arrays on graphs. The parameters for mixed covering arrays on graphs are given by a weighted graph. In this context, a weighted graph is a graph with a positive weight function $w : V(G) \to \mathbb{Z}^+$. Let G be a weighted graph with k vertices and weights $g_1 \leq g_2 \leq \ldots \leq g_k$, and let n be a positive integer. A *mixed covering array on G*, denoted by $CA(n, G, \prod_{i=1}^{k} g_i)$, is an $k \times n$ array with the following properties:

1. row i corresponds to a vertex $v_i \in V(G)$ with weight g_i;

2. the entries in row i are from $\mathbb{Z}_{g_i}$;

3. pair of rows which correspond to adjacent vertices of G are qualitatively independent.

Given a weighted graph G with weights $g_1, g_2, \ldots, g_k$, the mixed covering array number on G, denoted by $CAN(G, \prod_{i=1}^{k} g_i)$, is the minimum n for which there exists a $CA(n, G, \prod_{i=1}^{k} g_i)$. In 2007, Meagher *et al.* [67] and Cheng [24] studied the problem of constructing mixed covering arrays on graphs. Meagher, Moura, and Zekaoui [67] studied mixed covering arrays on graphs in detail and gave many powerful results including upper bounds on the mixed covering array number on all 3-chromatic and on a large number of 4- and 5-chromatic graphs. They built optimal mixed covering arrays on trees and cycles using some basic graph operations, and using a different technique, they built optimal mixed covering arrays on bipartite graphs. Cheng [24] mainly focused on algorithmic constructions for mixed covering arrays on graphs and on few families of hypergraphs and developed new techniques to construct optimal mixed covering arrays on bipartite graphs and cycles.

1.2.4 Variable strength covering arrays

In a software system, certain sets of parameters interact; certain sets of parameters are known not to interact. If prior knowledge of the system under testing indicates that certain parameters are known not to interact, it is unnecessary to test all interactions between them. A set of parameters that jointly affect on of the output values of a software system must be considered to interact. To model this situation, the notion of abstract simplicial complex *ASC* is used. The covering arrays on *ASCs* are called *variable strength covering arrays or covering arrays on hypergraphs*. The sets of parameters that we want to be tested are recorded as the facets of an *ASC*. Let Δ be an *ASC* over $\{0, \ldots, k-1\}$ with set of facets

Λ, and let $r = rank(\Delta)$. A variable strength covering array $VCA(n,\Lambda,g)$ is an $k \times n$ array over $\mathbb{Z}_g$ with rows $0,..,k-1$ such that if $\{b_0,..,b_{t-1}\} \in \Lambda$ for $t \leq r$, then these rows are t-qualitatively independent. The $VCAN(\Lambda,g)$ is defined to be the smallest n such that a $VCA(n,\Lambda,g)$ exists. Cohen *et al.* [29] used a simulated annealing algorithm to find *VCA* over a specific family of *ASCs* and presented several other related results. Cheng *et al.* [24] proposed a problem reduction technique involving proper hypergraph colouring and greedy algorithm to find *VCA* over arbitrary *ASC*. Mixed variable strength covering arrays have been systematically studied at length in Raaphorst's thesis [77]. He gave a complete solution for the problem of determining the covering array number and constructing optimal covering arrays on triangulation hypergraphs of the sphere.

1.2.5 Covering arrays with budget constraints

A practical limitation in the area of testing is the budget. Due to limited time, human, and computing resources, in most software developments, testing is performed with a fixed number of test cases. To model this situation, we consider the problem of building the best possible testing array within a fixed number of test cases, that is, fixed number of columns of the array. To test a software system with k parameters each having g values, the total number of t-tuples that needs to be covered for t-way interactions is $\binom{k}{t}g^t$. The *t-way configuration coverage* of a testing array $\mathscr{A}$ is defined by

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

where $N_t(\mathscr{A})$ is the number of distinct t-tuples covered in the columns of $\mathscr{A}$. Given fixed values of t,k,g and n, the problem is to build a testing array $\mathscr{A}$ of size at most n having high configuration coverage measure. This problem is called *covering array with budget constraints*. This is one of the five natural generalizations of covering arrays listed in [48]. A brief discussion of covering arrays with budget constraints problem is available in [48, 57] and [56, Chapter 7]. Maity [61] studied this problem when $t = 3$ and for specific values of g.

1.2.6 Covering arrays with forbidden configurations

For a set of t parameters, a *parameter-value configuration* is an ordered tuple of t valid values, one for each of the parameters. The generalization considered here applies to the situation in which some parameter-value configurations are invalid, a requirement quite common in software and hardware testing. If a system is susceptible to failure due to a single parameter-value configuration of two parameters, at least one of the tests given by a covering array is guaranteed to fail. These forbidden parameter-value configurations are modeled using a k-partite graph over $\sum_{i=1}^{k} g_i$ vertices while testing a system with k parameters and parameter i having g_i values for $1 \leq i \leq k$. These are called *covering arrays with forbidden configurations* and are studied by Danziger, Mendelsohn, Moura, and Stevens [36].

1.2.7 Covering arrays with column limit

Covering arrays with column limit, *CACLs*, are another generalization of covering arrays. A t-$CACL(n,k,g,w)$ is a $k \times n$ array with some empty cells. A parameter is represented by a row and takes values from $\mathbb{Z}_g$. In each column, there are exactly w non-empty cells, that is, there are w parameters that have values from $\mathbb{Z}_g$. The parameter w is called the *column limit*. Moreover, every $t \times n$ subarray contains each t-tuples from $\mathbb{Z}_g$ at least once. Covering arrays with column limit is useful in pharmacology where one has to limit the number of drugs administered to an individual at a time. Here a drug corresponds to a row and an individual corresponds to a column, and w is the number of drugs that can be administered to an individual at a time. Covering arrays with column limit are studied in [41]. The *CACLs* have close relation with *Group Divisible Covering Designs* [42].

1.3 Applications of covering arrays

Testing is an important but expensive part of software and hardware development process. Typically, more than 50% of the total development cost goes to software testing and verification. Software nonperformance and failure are expensive. There are many examples of the catastrophic impact of software and hardware failures. For example, a software failure interrupted the New York Mercantile Exchange and telephone service to several East Coast cities in February 1998 (Washington Technology, 1998) [79]. An example of hardware bug is the popular Pentium floating point division bug (1993) [39]. Because of this bug, the processor could return incorrect decimal results when dividing a number. This happened due to a flaw in the look up table employed in the division circuits and led to a loss of $475 million. A study by NIST shows that software errors cost U.S. economy $59.5 billion annually. The study also found that, although all errors cannot be removed, more than a third of these costs, or an estimated $22.2 billion, could be eliminated by an improved testing infrastructure [18]. Hence search for improved testing techniques has been a very active research area. We give some examples of testing problems to motivate the use of covering arrays. Covering arrays are useful in multiple applications, for example in software testing [1, 2, 52, 53, 56, 58], in hardware testing [47], and in drug screening [41, 48].

Software testing: Smartphones have become immensely popular because they combine communication capability with powerful graphical displays, internet browsing, and processing capability. A Huge number of smartphone applications, or *“apps”* are developed annually. An application, for example, Android apps, must operate across a variety of hardware and software platforms since not all products support the same options. For example, some smart phones may have three Keyboard options like `CrossTap`, `FlickKey` and `QWERTY`. The table below shows the names of five parameters for an application and values for each parameter.

Parameters	**Values**
Keyboard	CrossTap (0), FlickKey (1), QWERTY (2)
Navigation	DPAD (0), Trackball (1), Wheel (2)
Orientation	Landscape (0), Portrait (1), Square (2)
Screenlayout size	Large (0), Small (1), Normal (2)
Touchscreen	Finger (0), Notouch (1), Stylus (2)

An exhaustive test suite will include $3^5 = 243$ test configurations. However, only 11 test configurations are needed to cover all 2-way combinations of values. These 11 test cases are obtained using a 2-$CA(11,5,3)$ [75].

Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8	Case 9	Case 10	Case 11
CrossTap	CrossTap	CrossTap	FlickKey	QWERTY	FlickKey	QWERTY	FlickKey	QWERTY	FlickKey	QWERTY
DPAD	Trackball	Wheel	DPAD	DPAD	Wheel	Trackball	Trackball	Wheel	Trackball	Wheel
Landscape	Portrait	Square	Portrait	Square	Landscape	Landscape	Portrait	Square	Square	Portrait
Large	Normal	Small	Small	Normal	Normal	Small	Large	Large	Normal	Small
Finger	Notouch	Stylus	Notouch	Stylus	Notouch	Stylus	Stylus	Notouch	Finger	Finger

Using covering arrays, one can produce test suites that cover t-way combinations of values. For many applications, 2-way or 3-way testing may be appropriate, and either of these will require less than 1% of the total test cases required to cover all possible test configurations. Consider a system with 7 parameters having 4 values each. An exhaustive test suite will require $4^7 = 16384$ test cases. Table 1.1 shows the number of test cases required for t-way coverage at several values of t. This illustrates the power of covering arrays for combinatorial testing.

Some examples of case studies where covering arrays have been used efficiently are the following. Firstly, in Rich web application (RWA) [65], 2-way or pair-wise test found all but one fault found by exhaustive testing. It uses only 13% of the total test cases required for exhaustive testing. Secondly, in MP3 web application [106], most faults are detected by 2-way testing, except one fault that is caused by 4-way interaction. Finally, in web browser

DOM modules [71], all faults are detected using 4-way testing. It requires less than 5% of the total test cases required for exhaustive testing.

Table 1.1: Number of test cases required is a fraction of an exhaustive test suite.

t	Number of Test Cases [31]	% of Exhaustive
2	21	0.12
3	88	0.53
4	412	2.51
5	1536	9.37
6	4096	25.00

Hardware testing: Interaction testing is used for testing circuits and networks [98]. Consider a circuit with k inputs, each of one bit. Within the circuit, the input signals interact through arithmetic and logical operations to determine an output vector. This circuit can be exhaustively tested using 2^k tests. Like software, we expect errors to be revealed by a fraction of test configurations. Tang *et al.* [98], Borodai *et al.* [11] performed circuit testing in this environment using test configurations that cover t-way combinations of values. Seroussi and Bshouty gave a comprehensive study in [90]. In each case, a binary covering array of strength t is used to generate the test configurations.

Multiple-Drug-Therapy and Drug screening: In some cases, multiple drugs are taken simultaneously to treat a single disease. In such cases, interactions between drugs occur and cumulative effect of multiple drugs need to be studied before administering them. Another application of interaction testing is in drug screening [99]. Drug screening is a cost-effective method to quickly review all samples. Covering arrays help in establishing a rapid and effective method for drug screening.

Other applications of covering arrays include authentication [102], data compression [93], intersecting code [28], and universal hashing [19].

1.4 Thesis contributions

In this thesis, we consider the following four combinatorial problems related to covering arrays: An algebraic construction of strength four covering arrays; Testing arrays with high coverage measure; Covering arrays on product graphs; Mixed covering arrays on 3-uniform hypergraphs. We give below, chapter-wise, the problems considered and a brief outline of the solutions.

1. An Algebraic Construction of Strength Four Covering Arrays

This chapter focuses on constructing new strength four covering arrays and establishing improved bounds on the covering array numbers 4-$CAN(k,3)$. See also [8]. A strength four covering array 4-$CA(n,k,g)$, of size n, degree k, and order g, is a $k \times n$ array on g symbols such that every $4 \times n$ sub-array contains every 4×1 column on g symbols at least once. It is desirable in most applications to minimize the size n of covering arrays. The covering array number 4-$CAN(k,g)$ is the smallest n for which a 4-$CA(n,k,g)$ exists. There is no uniformly best algorithm for computing the smallest possible covering array for a particular problem. The method proposed here improves some of the best known upper bounds on the size of strength four covering arrays with $g = 3$.

Let $X = GF(g-1) \cup \{\infty\}$ be the set of g symbols on which we are to construct a 4-$CA(n,k,g)$. We choose g so that $g-1$ is a prime or prime power. Our construction called PGL construction, involves selecting a group G and finding vectors $u = (u_0, u_1, \ldots, u_{k-1})$, $v = (v_0, v_1, \ldots, v_{k-1}) \in X^k$, called starter vectors. We use the vectors to form a $k \times 2k$ matrix

M as follows:

$$M = \begin{pmatrix} u_0 & u_{k-1} & \dots & u_1 & v_0 & v_{k-1} & \dots & v_1 \\ u_1 & u_0 & \dots & u_2 & v_1 & v_0 & \dots & v_2 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ u_{k-2} & u_{k-3} & \dots & u_{k-1} & v_{k-2} & v_{k-3} & \dots & v_{k-1} \\ u_{k-1} & u_{k-2} & \dots & u_0 & v_{k-1} & v_{k-2} & \dots & v_0 \end{pmatrix}.$$

Let $G = PGL(2, g-1)$. For each $\sigma \in PGL(2, g-1)$, let M^σ be the matrix formed by the action of σ on the elements of M. The matrix obtained by developing M by G is the $k \times 2k|G|$ matrix $M^G = [M^\sigma : \sigma \in G]$. Let C be the $k \times g$ matrix that has a constant column with each entry equal to x, for each $x \in X$. Vectors $u, v \in X^k$ are said to be *starter vectors* for a 4-$CA(n,k,g)$ if any $4 \times 2k$ subarray of the matrix M has at least one representative from each non-constant orbit of $PGL(2, g-1)$ acting on 4-tuples from X. If starter vectors u, v exist in X^k (with respect to the group G) then $[M^G, C]$ is a 4-$CA(2kg(g-1)(g-2)+g, k, g)$. If we do not find vectors u and v, we look for vectors that produce an array with high 4-way configuration coverage. In order to complete the covering conditions, we add a small matrix C_1. In this case, $[M^G, C, C_1]$ forms a covering array. For some values of k, only one starter vector u and a C_1 matrix are enough to build a covering array. We use computer search to find starter vector(s) and matrix C_1.

We examine two methods, called *extending a solution* [69] and *randomized post optimization* [73], to obtain small improvements on the computational results obtained.

In the range of degrees considered in this chapter, the best known results previously come from [30]; in that paper, covering arrays are also found by using a group action on the symbols (the affine or Frobenius group), but no group action on the rows is employed. While for $g = 3$ the group that we employ on the symbols coincides with the affine group, we accelerate and improve the search by also exploiting a group action on the rows as in [21, 69], and develop a search method that can be applied effectively whenever $g \geq 3$ and $g-1$ is a prime power. PGL construction is an extension of the construction method developed in [21, 69]. The construction given in this chapter improves many of the current best known upper bounds on 4-$CAN(k,g)$ with $g=3$ and $19 \le k \le 74$.

2. Testing Arrays with High Coverage Measure

Using strength t covering arrays, one can generate test cases that cover t-way combinations of values. For most applications, 2-way (pair-wise) or 3-way testing may be effective [26, 61, 63] and either of these will require less than one percent of the time required to cover all possible test configurations. A major concern in the area of testing is the budget. Due to straightly limited time, human, and computing resources, in most software developments, testing is performed with a fixed number of test cases which may be significantly less than the number of test cases required even for 2-way or 3-way testing. To model this situation, we consider the problem of building the best possible testing array within a fixed number of test cases, that is, fixed number of columns of the array. To test a software system with k parameters each having g values, the total number of t-tuples that needs to be covered for t-way interactions is $\binom{k}{t}g^t$. The t-way configuration coverage of a testing array $\mathscr{A}$ is defined by

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

where $N_t(\mathscr{A})$ is the number of distinct t-tuples covered in the columns of $\mathscr{A}$. Given fixed values of t,k,g and $n,$ our objective is to build a testing array $\mathscr{A}$ of size at most n having high t-way configuration coverage. This is one of the five natural generalizations of covering arrays listed in [48]. This chapter presents algebraic constructions for testing arrays with high 3- or 4-way configuration coverage measure. See also [7, 6].

Given fixed values of k,n and $g,$ so that $g-2$ is a prime power, we are to construct a testing array $\mathscr{A}$ with high 3-way configuration coverage measure. Let $X = \{F_q, \infty_1, \infty_2\}$ be the set of g symbols (values) on which we are to construct a testing array having good configuration coverage $\mu_3(\mathscr{A})$. Clearly, $|X| = g = q+2$; we choose g so that $g-2$ is a prime or prime power. Our construction requires selecting a group G and finding a vector $v \in X^k$, called a vector with good configuration coverage measure. We use the vector $v = (v_0, v_1, \ldots, v_{k-1})$ to form a $k \times k$ circulant matrix M. The group acting on the matrix M

produces several matrices which are concatenated to form a testing array with high 3-way configuration coverage. The construction for a testing array with high 4-way configuration coverage is similar to the PGL construction described in Chapter 2.

The construction given here is an extension of the method developed by Meagher and Stevens in [69]. Their construction involves selecting a group $G < Sym_g$ and finding a vector $v \in \mathbb{Z}_g^k$, called a starter vector, to construct a strength two covering array of size $k|G| + g$ and numerous improved upper bounds were obtained for $CAN(2,k,g)$. Maity [61] generalizes this method to construct testing arrays with high 3-way configuration coverage measure for $g = p^m$ or $p^m + 1$ where p is a prime. This method produces several testing arrays with high 3-way configuration coverage measure about 95% to 99% for different values of g and k. In [61], a comparison of this method with tools like AETG [26] and IPOG [59] shows that this construction produces significantly smaller test suites.

Test coverage is one of the most important topics in software testing. Users would like to have some quantitative measure to judge the risk while using a product. Consider testing a software system with 40 parameters each having three values. Our construction for testing array with high 4-way configuration coverage generates a test suite with 243 test cases that ensure with probability 0.988 that the software cannot fail due to interactions of 2, 3 or 4 parameters whereas the best known covering array in [31] requires 465 test cases for full coverage. The results show that the proposed method could reduce the number of test cases significantly while compromising only slightly on the configuration coverage measure.

3. Covering Arrays on Product Graphs

The objects considered in this chapter are covering arrays on graphs and our primary concern is with constructions that make optimal covering arrays on large graphs that are obtained from a product of smaller graphs. See also [3]. A graph product is a binary operation on the set of all finite graphs. However, among all possible associative graph products, the most extensively studied in the literature are the Cartesian product, the direct product, the strong product and the lexicographic product. Here we recall the definition of Cartesian product only.

Definition : The *Cartesian product* of graphs G and H, denoted by $G\Box H$, is the graph with

$$V(G\Box H)=\{(g,h)|g\in V(G) \text{ and } h\in V(H)\},$$
$$E(G\Box H)=\{(g,h)(g',h')|g=g',hh'\in E(H), \text{ or } gg'\in E(G),h=h'\}.$$

The graphs G and H are called the *factors* of the product $G\Box H$.

In [68], the definition of a covering array has been extended to include a graph structure. This has been applied in the context of software testing by observing that we only need to test interactions between parameters that jointly affect one of the output values.
Two vectors x,y in $\mathbb{Z}_g^n$ are *qualitatively independent* if for all pairs $(a,b)\in\mathbb{Z}_g\times\mathbb{Z}_g$, there exists $i\in\{1,2,\ldots,n\}$ such that $(x(i),y(i))=(a,b)$. A covering array on a graph G, denoted by $CA(n,G,g)$, is a $|V(G)|\times n$ array on $\mathbb{Z}_g$ with the property that any two rows which correspond to adjacent vertices in G are qualitatively independent.
The smallest possible covering array on a graph G is denoted

$$CAN(G,g)=\min_{n\in\mathbb{N}}\big\{n\ :\ \text{there exists a } CA(n,G,g)\big\}.$$

Given a graph G and a positive integer g, a covering array on G with minimum size is called *optimal*. We start with a review of some definitions and results from product graphs in Section 4.3. In Section 4.5, we show that for all graphs G_1 and G_2,

$$\max_{i=1,2}\big\{CAN(G_i,g)\big\}\le CAN(G_1\Box G_2,g)\le CAN\big(\max_{i=1,2}\{\chi(G_i)\},g\big)$$

where $\chi(G_i)$ is the chromatic number of G_i. We look for graphs G_1 and G_2 where the lower bound on $CAN(G_1\Box G_2,g)$ is achieved. Let H be a finite group and S be a subset of $H\setminus\{id\}$ such that $S=-S$ (i.e., S is closed under inverse). The Cayley graph of H generated by S, denoted $Cay(H,S)$, is the undirected graph $G=(V,E)$ where $V=H$ and $E=\{(x,sx)\mid x\in H,s\in S\}$. In Section 4.6, we obtain families of Cayley graphs that achieve the lower bound on the covering array number on product graph. We list below two related theorems:

Theorem 4.6.2: Let H be a finite group and S be a generating set for H such that

1. $S = -S$ and $id \notin S$
2. $S^S = S$
3. there exist s_1 and s_2 in S such that $s_1 \neq s_2$ and $s_1 s_2 \in S$

then for $G_1 = Cay(H,S)$ and any three colourable graph G_2, we have

$$CAN(G_1 \Box G_2, g) = CAN(G_1, g).$$

Theorem 4.6.3: Let H be a finite group and S be a generating set for H such that

1. $S = -S$ and $id \notin S$
2. $S^S = S$
3. there exist s_1 and s_2 in S such that $s_1 \neq s_2$ and $s_1 s_2, s_1 s_2^{-1} \in S$

then for $G_1 = Cay(H,S)$ and any four colourable graph G_2, we have

$$CAN(G_1 \Box G_2, g) = CAN(G_1, g).$$

In Section 4.7, we present a polynomial time approximation algorithm with approximation ratio $\left\lceil \log\left(\frac{|V|}{2^{k-1}}\right)\right\rceil$ for constructing covering arrays on graphs $G = (V,E)$ having more than one prime factor with respect to the Cartesian product.

4. Mixed covering arrays on 3-uniform hypergraphs

Covering arrays have applications in many areas. Covering arrays are particularly useful in the design of test suites [26, 27, 47, 61, 62, 63]. The testing application is based on the following translation. Consider a software system that has k parameters, each parameter can take g values. Exhaustive testing would require g^k test cases for detecting software failure, but if k or g are reasonably large, this may be infeasible. We wish to build a test suite that tests all 3-way interactions of parameters with the minimum number of test cases. Covering arrays of strength 3 provide compact test suites that guarantee 3-way coverage of parameters.

To address different requirements of the software and hardware testing applications, many generalizations of covering arrays have been introduced (see [32, 48]). *Mixed covering arrays* are a generalization of covering arrays that allow different values for different rows. This fulfills the need that different parameters in the system take a different number of possible values. Some techniques to construct mixed covering arrays are presented in [34, 72]. Maegher and Stevens introduce and study covering array on graph in [68]. This is useful in testing applications where we may know in advance that two specific parameters do not interact. Then it is not necessary that each possible parameter-value configuration for these two parameters be tested, which allows reductions in the number of required test cases. We can use a graph structure to describe which pairs of parameters need to be tested. Serroussi and Bshouty [90] showed that finding an optimal covering array on a graph is NP-hard even for the binary case. Meagher, Moura, and Zekaoui [67] studied mixed covering arrays on graphs in details and gave many powerful results. Mixed variable strength covering arrays have been introduced and systematically studied at length in Raaphorst's thesis [77] and also dealt in [23].

The objects considered here generalize mixed covering arrays on graphs introduced in [67] but are a special case of mixed variable strength covering arrays introduced in [77], focusing on hypergraphs that are 3-uniform, rather than general hypergraphs. See also [5]. The motivation for this work is to improve applications of covering arrays to software, circuit and network systems. This also gives us new ways to study covering arrays construction.

Let n,k be positive integers with $k \geq 3$. Three vectors $x \in \mathbb{Z}_{g_1}^n$, $y \in \mathbb{Z}_{g_2}^n$, $z \in \mathbb{Z}_{g_3}^n$ are *3-qualitatively independent* if for any triplet $(a,b,c) \in \mathbb{Z}_{g_1} \times \mathbb{Z}_{g_2} \times \mathbb{Z}_{g_3}$, there exists an index $j \in \{1,2,...,n\}$ such that $(x(j),y(j),z(j)) = (a,b,c)$. Let H be a 3-uniform hypergraph with k vertices $v_1, v_2, \ldots, v_k$ with respective vertex weights $g_1, g_2, \ldots, g_k$. A mixed covering array on H, denoted by $CA(n,H,\prod_{i=1}^{k} g_i)$, is a $k \times n$ array such that row i corresponds to vertex v_i, entries in row i are from Z_{g_i}; and if $\{v_x, v_y, v_z\}$ is a hyperedge in H, then the rows x, y, z are 3-qualitatively independent. The parameter n is called the size of the array. Given a weighted 3-uniform hypergraph H, a mixed covering array on H with minimum size is called optimal.

In Section 5.1, we outline the necessary background in the theory of hypergraphs. In Section 5.2, we recall the definition of mixed covering arrays on hypergraphs and related results. In Section 5.3, we give results related to balanced and pairwise balanced vectors which are required for basic hypergraph operations.

Definition: A length-n vector with alphabet size g is *balanced* if each symbol occurs $\lfloor n/g \rfloor$ or $\lceil n/g \rceil$ times.

Definition: Two length-n vectors x_1 and x_2 with alphabet size g_1 and g_2 are *pairwise balanced* if both vectors are balanced and each pair of symbols $(a,b) \in \mathbb{Z}_{g_1} \times \mathbb{Z}_{g_2}$ occurs $\lfloor n/g_1 g_2 \rfloor$ or $\lceil n/g_1 g_2 \rceil$ times in (x_1, x_2), so for $n \geq g_1 g_2$ pairwise balanced vectors are always 2-qualitatively independent.

Definition: Let H be a weighted hypergraph. A *balanced covering array* on H is a covering array on H in which every row is balanced and the rows correspond to vertices in a hyperedge are pairwise balanced.

In this section, we prove two important theorems related to the construction of optimal mixed covering arrays on some specific class of 3-uniform hypergraphs. Theorem 5.3.2 proves Conjecture 3.4.27 posted by Raaphorst in [77].

Theorem 5.3.1: Let $x_1 \in \mathbb{Z}_{g_1}^n$ and $x_2 \in \mathbb{Z}_{g_2}^n$ be two balanced vectors. Then for any positive integer h, there exists a balanced vector $y \in \mathbb{Z}_h^n$ such that x_1 and y are pairwise balanced and

x_2 and y are pairwise balanced.

Theorem 5.3.2: Let $x_1 \in \mathbb{Z}_{g_1}^n$ and $x_2 \in \mathbb{Z}_{g_2}^n$ be two pairwise balanced vectors. Then for any h such that $g_1 g_2 h \leq n$, there exists a balanced vector $y \in \mathbb{Z}_h^n$ such that x_1, x_2 and y are 3-qualitatively independent and x_1 and y are pairwise balanced and x_2 and y are pairwise balanced.

In Section 5.4, we introduce four basic hypergraph operations:

1. *Single-vertex edge hooking I*
2. *Single-vertex edge hooking II*
3. *Two-vertex hyperedge hooking*
4. *Single-vertex hyperedge hooking*

Using these operations, we construct optimal mixed covering arrays on α-acyclic 3-uniform hypergraphs, 3-uniform interval hypergraphs, conformal 3-uniform hypertrees having a binary tree as host tree, 2-tree hypergraphs, and 3-uniform loose cycles. In this section, we give a solution to Conjecture 3.4.28 posted by Raaphorst in [77].

Chapter 2

An Algebraic Construction of Strength Four Covering Arrays

Covering arrays are useful in multiple applications, for example in software testing [1, 2, 47, 52, 53, 56, 58, 70], in experimental designs [49, 82, 97] and in drug screening [41, 48]. Pair-wise (2-way) interaction testing requires that for a given number of input parameters to the system, each possible combination of values for any pair of parameters be covered by at least one test case. 2-way and 3-way interaction testing are known to be effective for different types of software testing [26, 61, 63]. In the real world, there may be 4, 5 or even more, parameters involved in failures, so our test suite covering 2-way and 3-way interactions might not detect them. Depending on the budget and the software, two-way through five-way or six-way interaction testing may be appropriate [58]. Here we consider the problem of constructing strength four covering arrays. Each column of a strength four covering array is a test case for the problem of 4-way interaction testing of parameters.

In this chapter, we present an algebraic construction that improves many of the best known upper bounds on n for covering arrays 4-$CA(n,k,g)$ with $g = 3$. See also [8]. In Section 2.1, we summarize the results from group theory that we use in our construction. In Section 2.2, we describe two known constructions for covering arrays. In Section 2.3, we give a new construction that can be used to build covering arrays 4-$CA(n,k,g)$ for $g = p^m + 1$, p is prime. This construction is an extension of the construction method developed in [21, 69]. The construction given in this chapter improves many of the current best known upper bounds on 4-$CAN(k,g)$ with $g = 3$ and $19 \leq k \leq 74$. Colbourn [31] maintains a repository of the best known upper bounds for covering array sizes. In Section 2.4, we examine two methods to obtain small improvements on the computational results obtained. In Section 2.5, we present the computational results.

2.1 Transitive action of groups

In this section, we summarize the results from group theory that we use. All the definitions and results mentioned in this section are standard and referred from [81]. A *permutation* of a non-empty set X is a bijection $\pi : X \to X$. The set of all permutations of X, denoted by *Sym* X, forms a group with respect to functional composition. The set *Sym* X of all permutations of X is called the *symmetric group*. If X is a non-empty set, a subgroup G of the symmetric group *Sym* X is called a *permutation group*. The *degree* of a permutation group is the cardinality of X.

Two elements x and y of X are said to be *equivalent* under G if there exists a permutation π in G that maps x to y. This relation is an equivalence relation on X and the equivalence classes are called *orbits* under G. More formally, for each $x \in X$, let

$$\text{orbit}_G(x) = \{x\pi \mid \pi \in G\}.$$

The set $\text{orbit}_G(x)$ is a subset of X called the orbit of x under G. For each $x \in X$, let

$$\text{stab}_G(x) = \{\pi \in G \mid x\pi = x\}.$$

We call the set $\text{stab}_G(x)$ the *stabilizer* of x in G. The permutation group G is called *transitive* if, given any pair of elements x and y of X, there exists a permutation π in G which maps x to y. Thus G is transitive if and only if there is exactly one orbit under G, which is X itself.

Let the cardinality of X be n and G be a permutation group on X. For $1 \leq k \leq n$, we denote $X^{[k]}$ for the set of all ordered k-tuples of distinct elements $(x_1, x_2, \ldots, x_k)$. The permutation group G exhibits natural action component wise on $X^{[k]}$, that is, if $\pi \in G$,

$$(x_1, x_2, \ldots, x_k)\pi \rightarrow (x_1\pi, x_2\pi, \ldots, x_k\pi).$$

Definition 2.1.1. A permutation group G on X is said to be *k-transitive on X* if for any two ordered k-tuples of distinct elements $(x_1, x_2, \ldots, x_k)$ and $(y_1, y_2, \ldots, y_k)$ in $X^{[k]}$ there is a $\pi \in G$ such that $x_i\pi = y_i$ for $1 \leq i \leq k$.

Definition 2.1.2. A permutation group G on X is said to be *sharply k-transitive on X* if given two tuples in $X^{[k]}$, there exists a unique permutation in G mapping one k-tuple to the other.

Theorem 2.1.1. *1. The symmetric group S_n is sharply n-transitive.*

2. If $n > 2$, the alternating group A_n is sharply $(n-2)$-transitive.

There are certain examples of sharply 2-transitive and 3-transitive permutation groups that are not of alternating or symmetric type. Some examples are fractional linear group $L(q)$ or projective general linear group $PGL(2,q)$ and its subgroup called affine linear group $AGL(1,q)$.

2.1.1 Fractional linear group

Let F_q be a Galois field $GF(q)$ where $q = p^m$ and p is a prime. We now adjoin a new element, which we denote by ∞, to F_q to obtain a set $X = F_q \cup \{\infty\}$. One may think of the resulting set $X = F_q \cup \{\infty\}$ as the projective line consisting of $q+1$ points. Define

$$L(q) = \left\{ \sigma : X \rightarrow X \mid x\sigma = \frac{ax+b}{cx+d}, \text{ where } a,b,c,d \in F_q \text{ and } ad-bc \neq 0 \right\}$$

with standard convention about ∞, for example, $x + \infty = \infty$, $x \times \infty = \infty$, $\frac{\infty}{\infty} = 1$, $\frac{x}{0} = \infty$ for $x \neq 0$, and $\frac{a\infty+c}{b\infty+c} = \frac{a}{b}$.

It is easy to verify that $L(q)$ is a group with respect to functional composition, called *fractional linear group*. In fact, $L(q)$ is isomorphic with the *projective general linear group* $PGL(2,q)$. Note that

$$|L(q)| = |PGL(2,q)| = \frac{(q^2-1)(q^2-q)}{q-1} = (q+1)q(q-1).$$

We now define

$$H(q) = \left\{\alpha : X \to X \mid x\alpha = ax+b \text{ where } a,b \in F_q \text{ and } a \neq 0\right\}.$$

Note that $H(q)$ is the stabilizer of ∞ in $L(q)$ and hence a subgroup of $L(q)$. The group $H(q)$ is isomorphic to affine general linear group $AGL(1,q)$ and $|H(q)| = q(q-1)$.

Theorem 2.1.2. *The group $H(q)$ is sharply 2-transitive on $GF(q)$ with degree q. The group $L(q)$ is sharply 3-transitive on $GF(q) \cup \{\infty\}$ with degree $q+1$.*

For the undefined terms and more details see [81, Chapter 7].

2.2 Some known constructions

There are several known methods for constructing covering arrays. In this section, we review two constructions: the finite field construction (Bush 1952) [17, 47] and algebraic constructions [21, 60, 69].

2.2.1 The finite field construction

It is well-known that there exists a set of $g-1$ mutually orthogonal Latin squares of order g if and only if there exists a finite projective plane of order g. It is also well-known that a finite projective plane exists when the order g is a power of a prime, that is $g = p^m$ for $m \geq 1$. The construction of projective planes of prime order was generalized by Bush [17]. Theorem 2.2.1 ensures existence of orthogonal array $OA(g^t, g+1, g, t)$ when g is a prime power, which are also t-$CA(g^t, g+1, g)$.

Theorem 2.2.1. [17] Let $g = p^m$ be a prime power with $g > t$. Then t-$CAN(g+1, g) = g^t$.

Proof. The proof given here is similar to that given in [47]. Let $F = \{0, 1, \dots\}$ be the finite field of order g, with 0 being the zero element of the field. We construct a $g+1 \times g^t$ array A whose rows are indexed by members of $F \cup \{\infty\}$ and whose columns are indexed by t-tuples $(\alpha_0, \alpha_1, \dots, \alpha_{t-1}) \in F^t$. The entry in this array in the row indexed x $(\neq 0)$ and column indexed $(\alpha_0, \alpha_1, \dots, \alpha_{t-1})$ is defined to be $\sum_{j=0}^{t-1} \alpha_j x^j$. The entry in this array in the row ∞ (resp. 0) and column indexed $(\alpha_0, \alpha_1, \dots, \alpha_{t-1})$ is defined to be α_0 (resp. α_{t-1}). Now consider $T = (f_1, f_2, \dots, f_{t-1})$ an arbitrary t-tuples of F^t. Let A' be a submatrix of A induced by an arbitrary choice of t rows $(x_1, x_2, \dots, x_t)$. To complete the proof we need to show that T is a column of A'. To verify this we first consider the case when $x_i \neq 0, \infty$ for $1 \leq i \leq t$. We solve the following system of t equations for t unknown quantities α_j, which index the column T:

$$\sum_{j=0}^{t-1} \alpha_j x_i^j = f_i \qquad \text{with } 1 \leq i \leq t.$$

Note that the system of equations has a unique solution as the coefficient matrix has the form of a Vandermonde matrix, which is invertible. Secondly, suppose that the submatrix A' contains either row 0 or ∞ or both, then we have a system of $t-1$ or $t-2$ equations that also have a Vandermonde coefficient matrix, and thus has a unique solution. $\square$

Example 2.2.1. Below is an example of a 2-$CA(25, 6, 5)$ obtained using the construction given in Theorem 2.2.1.

	00	01	02	03	04	10	11	12	13	14	20	21	22	23	24	30	31	32	33	34	40	41	42	43	44
0	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
1	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
2	0	2	4	1	3	1	3	0	2	4	2	4	1	3	0	3	0	2	4	1	4	1	3	0	2
3	0	3	1	4	2	1	4	2	0	3	2	0	3	1	4	3	1	4	2	0	4	2	0	3	1
4	0	4	3	2	1	1	0	4	3	2	2	1	0	4	3	3	2	1	0	4	4	3	2	1	0
∞	0	0	0	0	0	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3	4	4	4	4	4

2.2.2 Algebraic constructions

Chateauneuf, Colbourn and Kreher [20] introduced an algebraic method to construct covering arrays of strength three. The idea is to construct a covering array starting from a small array, a *starter array M,* and a group G. The goal is to choose the matrix M and group G so that the group acting on the array M produces several arrays which are concatenated to form a covering array. In some cases, a small array will be appended to complete the covering condition. For example, when $G = Sym\{0,1,2\}$ and

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

we get a covering array 3-$CA(27,4,3)$ as shown below:

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 2 & 0 & 0 & 2 & 1 & 2 & 2 & 1 & 0 & 1 & 1 & 1 & 2 & 1 & 1 & 2 & 0 & 2 & 2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 2 & 2 & 2 & 2 & 0 & 0 \\ 1 & 0 & 2 & 0 & 2 & 0 & 1 & 0 & 1 & 2 & 0 & 2 & 0 & 1 & 2 & 1 & 2 & 1 & 0 & 1 & 0 & 2 & 1 & 2 \\ 2 & 1 & 2 & 1 & 1 & 2 & 1 & 2 & 0 & 1 & 0 & 1 & 2 & 0 & 2 & 0 & 0 & 2 & 0 & 2 & 1 & 0 & 1 & 0 \end{bmatrix}$$

An array is a starter array for $t = 3$ if on each set of three rows there is a representative from each non-constant orbit of G acting on 3-tuples from the set of symbols X. They used one factorization of complete graphs, combinatorial designs like near resolvable design and pairs of disjoint Steiner triple systems in order to construct starter array M.

Meagher and Stevens [69] extended the idea of Chateauneuf, Colbourn and Kreher and proposed a strategy for construction of covering arrays of strength two, and several improved upper bounds were obtained for $CAN(k,g)$. A key advantage of their method is that they search for a small vector, called *starter vector*, that is used to construct a starter array, and hence a covering array. This construction involves selecting a subgroup of the symmetric group on g elements, $G < Sym_g$, and finding a starter vector $v \in \mathbb{Z}_g^k$. The starter vector depends on the group G. They used $G = \langle(1, 2, \ldots, g-1)\rangle < Sym_g$. Note that G employs action on $g-1$ symbols and fixes one symbol 0. The vector v is used to form a circulant array M. The vector v is selected such that M is a starter array, that is, on each set of two rows there is a representative from each non-constant orbit of G acting on 2-tuples from $\mathbb{Z}_g$. The group G acting on M produces several matrices which are concatenated to for a covering array. Often, it is needed to add a small matrix, to complete the covering conditions. Using this construction, if a starter vector exists in $\mathbb{Z}_g^k$ with respect to G, then there exists a $CA\,(k(g-1)+1, k, g)$.

Finally, Lobb, Colbourn, Danziger, Stevens and Torres-Jimenez [60] extended the idea of Meagher and Stevens by permitting the action of the group on the symbols to fix f symbols, where f is any non-negative integer and by allowing the group to be an arbitrary group of order $g-f$. When the number of fixed symbols can take any non-negative value f, it suffices to use a group of order $g-f$, thereby requiring only $g-f$ matrices to be concatenated to construct a covering array. This construction demonstrated improvements in upper bounds for numerous covering array numbers of strength two using heuristic search.

2.3 PGL construction

The construction given in this section is new and improves many of the upper bounds on the size of strength four covering arrays. Let $X = GF\,(g-1) \cup \{\infty\}$ be the set of g symbols on which we are to construct a 4-$CA(n, k, g)$. We choose g so that $g-1$ is a prime or prime power.

2.3.1 Case 1: Two starter vectors

Our construction involves selecting a group G and finding vectors $u, v \in X^k$, called starter vectors. We use the vectors to form a $k \times 2k$ matrix M.

$$M = \begin{pmatrix} u_0 & u_{k-1} & \cdots & u_1 & v_0 & v_{k-1} & \cdots & v_1 \\ u_1 & u_0 & \cdots & u_2 & v_1 & v_0 & \cdots & v_2 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ u_{k-2} & u_{k-3} & \cdots & u_{k-1} & v_{k-2} & v_{k-3} & \cdots & v_{k-1} \\ u_{k-1} & u_{k-2} & \cdots & u_0 & v_{k-1} & v_{k-2} & \cdots & v_0 \end{pmatrix}.$$

Let $G = PGL(2, g-1)$. For each $\sigma \in PGL(2, g-1)$, let M^σ be the matrix formed by the action of σ on the elements of M. The matrix obtained by developing M by G is the $k \times 2k|G|$ matrix $M^G = [M^\sigma : \sigma \in G]$. Let C be the $k \times g$ matrix that has a constant column with each entry equal to x, for each $x \in X$. Vectors $u, v \in X^k$ are said to be *starter vectors* for a 4-$CA(n, k, g)$ if any $4 \times 2k$ subarray of the matrix M has at least one representative from each non-constant orbit of $PGL(2, g-1)$ acting on 4-tuples from X. Under this group action, there are precisely $g + 11$ orbits of 4-tuples. These $g + 11$ orbits are determined by the pattern of entries in their 4-tuples:

1. $\{(a,a,a,a)^T : a \in X\}$
2. $\{(a,a,a,b)^T : a,b \in X, a \neq b\}$
3. $\{(a,a,b,a)^T : a,b \in X, a \neq b\}$
4. $\{(a,b,a,a)^T : a,b \in X, a \neq b\}$
5. $\{(b,a,a,a)^T : a,b \in X, a \neq b\}$
6. $\{(a,a,b,b)^T : a,b \in X, a \neq b\}$
7. $\{(a,b,a,b)^T : a,b \in X, a \neq b\}$

8. $\{(a,b,b,a)^T : a,b \in X, a \neq b\}$

9. $\{(a,a,b,c)^T : a,b,c \in X, a \neq b \neq c\}$

10. $\{(b,a,a,c)^T : a,b,c \in X, a \neq b \neq c\}$

11. $\{(a,b,a,c)^T : a,b,c \in X, a \neq b \neq c\}$

12. $\{(b,a,c,a)^T : a,b,c \in X, a \neq b \neq c\}$

13. $\{(a,b,c,a)^T : a,b,c \in X, a \neq b \neq c\}$

14. $\{(b,c,a,a)^T : a,b,c \in X, a \neq b \neq c\}$

15. $g-3$ orbits of patterns with four distinct entries. The reason is this. There are $g(g-1)(g-2)(g-3)$ 4-tuples with four distinct entries and each orbit contains $g(g-1)(g-2)$ 4-tuples as $|PGL(2,g-1)| = g(g-1)(g-2)$.

If starter vectors u,v exist in X^k (with respect to the group G) then there exists a 4-$CA(2kg(g-1)(g-2)+g,k,g)$. We give an example to explain the method.

Example 2.3.1. Let $g=3$, $k=30$, $X = GF(2)\cup\{\infty\}$ and $G = PGL(2,2)$. The action of G on 4-tuples from X has 14 orbits:

Orb 1: $\{(0,0,0,0)^T, (\infty,\infty,\infty,\infty)^T, (1,1,1,1)^T\}$

Orb 2: $\{(0,0,0,1)^T, (0,0,0,\infty)^T, (\infty,\infty,\infty,0)^T, (\infty,\infty,\infty,1)^T, (1,1,1,0)^T, (1,1,1,\infty)^T\}$

Orb 3: $\{(1,\infty,\infty,\infty)^T, (1,0,0,0)^T, (0,1,1,1)^T, (\infty,0,0,0)^T, (0,\infty,\infty,\infty)^T, (\infty,1,1,1)^T\}$

Orb 4: $\{(0,1,0,0)^T, (\infty,0,\infty,\infty)^T, (0,\infty,0,0)^T, (\infty,1,\infty,\infty)^T, (1,0,1,1)^T, (1,\infty,1,1)^T\}$

Orb 5: $\{(1,1,\infty,1)^T, (\infty,\infty,1,\infty)^T, (0,0,1,0)T, (1,1,0,1)^T, (0,0,\infty,0)^T, (\infty,\infty,0,\infty)^T\}$

Orb 6: $\{(1,1,\infty,\infty)^T, (\infty,\infty,1,1)^T, (0,0,1,1)^T, (1,1,0,0)^T, (0,0,\infty,\infty)^T, (\infty,\infty,0,0)^T\}$

Orb 7: $\{(\infty,0,\infty,0)^T, (0,1,0,1)^T, (\infty,1,\infty,1)^T, (0,\infty,0,\infty)^T, (1,0,1,0)^T, (1,\infty,1,\infty)^T\}$

Orb 8: $\{(\infty,1,1,\infty)^T,(1,\infty,\infty,1)^T,(1,0,0,1)^T,(0,1,1,0)^T,(\infty,0,0,\infty)^T,(0,\infty,\infty,0)^T\}$

Orb 9: $\{(1,1,\infty,0)^T,(\infty,\infty,1,0)^T,(0,0,1,\infty)^T,(1,1,0,\infty)^T,(0,0,\infty,1)^T,(\infty,\infty,0,1)^T\}$

Orb 10: $\{(\infty,0,\infty,1)^T,(0,1,0,\infty)^T,(\infty,1,\infty,0)^T,(0,\infty,0,1)^T,(1,0,1,\infty)^T,(1,\infty,1,0)^T\}$

Orb 11: $\{(1,\infty,0,1)^T,(0,\infty,1,0)^T,(\infty,1,0,\infty)^T,(0,1,\infty,0)^T,(\infty,0,1,\infty)^T,(1,0,\infty,1)^T\}$

Orb 12: $\{(1,\infty,0,\infty)^T,(0,\infty,1,\infty)^T,(\infty,1,0,1)^T,(0,1,\infty,1)^T,(\infty,0,1,0)^T,(1,0,\infty,0)^T\}$

Orb 13: $\{(1,\infty,0,0)^T,(0,\infty,1,1)^T,(\infty,1,0,0)^T,(0,1,\infty,\infty)^T,(\infty,0,1,1)^T,(1,0,\infty,\infty)^T\}$

Orb 14: $\{(1,\infty,\infty,0)^T,(1,0,0,\infty)^T,(0,1,1,\infty)^T,(\infty,0,0,1)^T,(0,\infty,\infty,1)^T,(\infty,1,1,0)^T\}$

The following are starter vectors to construct $[M^G,C]$, a 4-$CA(363,30,3)$:

$$u = (011\infty11\infty\infty\infty001\infty\infty\infty1\infty10\infty\infty0\infty1100\infty01)$$

$$v = (11\infty\infty01101000\infty101\infty1\infty0\infty000010\infty\infty\infty).$$

We used computer search to find u and v. One can check that on each set of 4 rows of M there is a representative from each orbit 2-14. Thus, 4-$CAN(30,3) \leq 363$.

2.3.2 Choice of starter vectors u and v

The problem is to find two vectors $u,v \in X^k$ such that on each set of 4 rows of M there is a representative from each orbit 2-15. To determine which vectors work as starters, we define the sets $d[x,y,z]$ for positive integers x,y and z as follows:

$$d[x,y,z] = \Big\{(u_i,u_{i+x},u_{i+x+y},u_{i+x+y+z}) : 0 \leq i \leq k-1\Big\} \bigcup \Big\{(v_i,v_{i+x},v_{i+x+y},v_{i+x+y+z}) : 0 \leq i \leq k-1\Big\}$$

where the subscripts are taken modulo k. For computational convenience, we partition the collection of $\binom{k}{4}$ choices of four distinct rows from k rows into disjoint equivalence classes.

Formally, let S be the set of all $\binom{k}{4}$ 4-combinations of the set $\{0,1,\ldots,k-1\}$. Define a binary relation R on S by putting

$$\{s_1, s_2, s_3, s_4\}\ R\ \{s_1', s_2', s_3', s_4'\} \text{ iff}$$

$$\{s_1 + d, s_2 + d, s_3 + d, s_4 + d\} = \{s_1', s_2', s_3', s_4'\} \text{ for some } d \in \mathbb{N}$$

where all of the addition is modulo k. Because R is an equivalence relation on S, S can be partitioned into disjoint equivalence classes. The equivalence class determined by $\{s_1, s_2, s_3, s_4\} \in S$ is given by

$$[\{s_1, s_2, s_3, s_4\}] = \Big\{\{s_1 + d, s_2 + d, s_3 + d, s_4 + d\} | 0 \le d \le k-1\Big\}.$$

Without loss of generality, we may assume that $0 = s_1 < s_2 < s_3 < s_4$ for each equivalence class representative $[\{s_1, s_2, s_3, s_4\}]$. As an illustration, when $X = \{0, 1, 2, ..., 7\}$. S is partitioned into 10 disjoint equivalence classes:

$$[\{0,1,2,3\}] \quad [\{0,1,2,4\}] \quad [\{0,1,2,5\}] \quad [\{0,1,2,6\}] \quad [\{0,1,3,4\}]$$

$$[\{0,1,3,5\}] \quad [\{0,1,3,6\}] \quad [\{0,1,4,5\}] \quad [\{0,1,4,6\}] \quad [\{0,2,4,6\}]$$

A distance vector (x, y, z, w) is associated with every equivalence class $[\{s_1, s_2, s_3, s_4\}]$ where $x = s_2 - s_1$, $y = s_3 - s_2$, $z = s_4 - s_3$, $w = s_1 - s_4 \bmod k$. The fourth distance is redundant because $x + y + z + w = k$. We rewrite the equivalence class of 4-combinations $[\{s_1, s_2, s_3, s_4\}]$ as

$$[x, y, z] = \Big\{\{i, i + x, i + x + y, i + x + y + z\} | i = 0, 1, 2, ..., k-1\Big\}.$$

For $k = 8$, $[1,1,1] = [\{0,1,2,3\}]$, $[1,1,2] = [\{0,1,2,4\}]$, $[1,1,3] = [\{0,1,2,5\}]$, $[1,1,4] = [\{0,1,2,6\}]$, $[1,2,1] = [\{0,1,3,4\}]$, $[1,2,2] = [\{0,1,3,5\}]$, $[1,2,3] = [\{0,1,3,6\}]$, $[1,3,1] = [\{0,1,4,5\}]$, $[1,3,2] = [\{0,1,4,6\}]$, $[2,2,2] = [\{0,2,4,6\}]$.

Lemma 2.3.1. *Let S be the set of all 4-combinations of $\{0, 1, \ldots, k-1\}$. Then S can be partitioned into disjoint equivalence classes*

$$[x, y, z] = \Big\{\{i, i + x, i + x + y, i + x + y + z\} | i = 0, 1, 2, ..., k-1\Big\}$$

where $x = 1, 2, ..., \lfloor \frac{k}{4} \rfloor$, $y = x, x+1, ..., k-1$ and $z = x, x+1, ..., k-1$ such that *(i)* $2x+y+z<k$ *when* $z>x$.

(ii) $x \le y \le \lfloor \frac{k-2x}{2} \rfloor$ *when* $z=x$.

There are no further classes distinct from these.

Before proving the result, we give an example. When S is the set of all 4-combinations of $\{0,1,2,3,4,5,6,7\}$, S can be partitioned into 10 disjoint classes: $[1,1,1]$, $[1,1,2]$, $[1,1,3]$, $[1,1,4]$, $[1,2,1]$, $[1,2,2]$, $[1,2,3]$, $[1,3,1]$, $[1,3,2]$ and $[2,2,2]$.

Proof. Let (x,y,z,w) be the distance vector corresponding to equivalence class $[\{s_1,s_2,s_3,s_4\}]$ where $x=s_2-s_1$, $y=s_3-s_2$, $z=s_4-s_3$, $w=s_1-s_4$ (mod k). Then,

$$[\{s_1,s_2,s_3,s_4\}] = [x,y,z] = [y,z,w] = [z,w,x] = [w,x,y]. \tag{2.1}$$

Without loss of generality, we choose $[x,y,z]$ as class representative if $x \le y$, $x \le z$ and $x \le w$. Thus $1 \le x \le \frac{k}{4}$, $y = x, x+1, \ldots, k-1$ and $z = x, x+1, \ldots, k-1$. We consider two cases.

Case (i) $z>x$ and $x \le y \le k-1$. Here we prove that w has to be strictly greater than x. If $w=x$ and $y>x$, then Equation 2.1 gives $[x,y,z]=[x,x,y]$. But classes of the form $[a,a,b]$ are also generated when $x=y$. If $w=x$ and $y=x$, then Equation 2.1 gives $[x,x,z]=[x,x,x]$. But the classes of the form $[a,a,a]$ are also generated under Case (ii). Therefore, in order to avoid repetition, w has to be strictly greater than x. That is, $w=k-x-y-z>x$ which implies $2x+y+z<k$.

Case (ii): $z=x$ and $x \le y \le k-1$. If $z=x$, then Equation 2.1 gives $[x,y,x]=[x,w,x]$; they are obtained from the distance vector (x,y,x,w) where $y+w=k-2x$. Thus it suffices to consider the classes of the form $[x,y,x]$ for $y \le \lfloor \frac{k-2x}{2} \rfloor$ only. Hence the lemma follows. □

At this stage, we make a few remarks about the size of equivalence classes defined by above choices of x,y and z.

1. If k is an odd integer, each class contains exactly k distinct choices from the collection of $\binom{k}{4}$ choices and hence there are $l = \frac{(k-1)(k-2)(k-3)}{24}$ distinct classes of size k.

2. If k is an even integer, $\frac{k}{2}$ can be written as sum of two positive integers a and b where $a \le b$ in $\lfloor \frac{k}{4} \rfloor$ different ways.

   Case 1: If $k \not\equiv 0$ (mod 4), a class of the form $[a,b,a]$ contains only $\frac{k}{2}$ distinct choices. There are total $\lfloor \frac{k}{4} \rfloor$ equivalence classes of the form $[a,b,a]$ with size $\frac{k}{2}$ and the remaining classes are of size k.

   Case 2: If $k \equiv 0$ (mod 4), a class of the form $[a,b,a]$ contains only $\frac{k}{2}$ distinct choices and a class of the form $[a,a,a]$ where $a = \frac{k}{4}$ contains only $\frac{k}{4}$ distinct choices. Here we get total $\frac{k}{4} - 1$ equivalence classes of size $\frac{k}{2}$, exactly one class of size $\frac{k}{4}$ and the remaining classes are of size k.

For $k = 8$, there are 10 equivalence classes. The classes $[1,3,1]$ and $[2,2,2]$ are of size 4 and 2 respectively and the remaining 8 classes are of size 8 each. Thus $8 \times 8 + 4 + 2 = \binom{8}{4}$.

Algorithm 1 will generate all the equivalence classes without repetition.

Theorem 2.3.1. *Let $X = GF(g-1) \cup \{\infty\}$ and $G = PGL(2, g-1)$. If there exists a pair of vectors $u, v \in X^k$ such that each $d[x,y,z]$ has a representative from each of the orbits 2-15, then there exists a 4-$CA(2kg(g-1)(g-2)+g,k,g)$ covering array.*

Proof. Let $u, v \in X^k$ be vectors such that each $d[x,y,z]$ has a representation from each of the orbits 2-15. Using u, v, we create the matrix $[M^G, C]$. Let $\{s_1, s_2, s_3, s_4\}$ be a member in S. By Lemma 1, there exists three positive integers x_0, y_0 and z_0 such that $\{s_1, s_2, s_3, s_4\} \in [x_0, y_0, z_0]$. It is given that $d[x_0, y_0, z_0]$ has a representative from each of the orbits 2-15. In other words, if we look at the rows s_1, s_2, s_3, s_4 of M, we see a representative from each of the orbits 2-15. Consequently, because $PGL(2, g-1)$ is 3-transitive on X, $[M^G, C]$ is a 4-$CA(2kg(g-1)(g-2)+g,k,g)$. □

Algorithm 1 Equivalence-Classes($k, 4$)

```
Input: $k$
Output: All $[x,y,z]$ classes.
for $x \leftarrow 1$ to $\frac{k}{4}$ do
  for $y \leftarrow x$ to $k-1$ do
    if $y > \frac{k-2x}{2}$ then
      for $z \leftarrow x+1$ to $k-2x-y-1$ do
        add $[x,y,z]$
      end for
    else
      if $y == \frac{k-2x}{2}$ and $x == \frac{k-2x}{2}$ then
        add $\left[\frac{k}{4}, \frac{k}{4}, \frac{k}{4}\right]$
      else
        for $z \leftarrow x$ to $k-2x-y-1$ do
          add $[x,y,z]$
        end for
      end if
    end if
  end for
end for
```

2.3.3 Case 2: Two vectors u, v and a matrix C_1

If we do not find vectors u and v such that each $d[x,y,z]$ contains a representative from each of the orbits 2-15, we look for vectors that produce an array with maximum possible coverage. In order to complete the covering conditions, we add a small matrix C_1. We give an example below to illustrate the technique.

Example 2.3.2. Let $k = 21$ and $g = 3$. Here we are unable to find vectors u and v such that each $d[x,y,z]$ contains a representative from each of the orbits 2-15. For $k = 21$, there are 285 $[x,y,z]$ classes. All classes $[x,y,z]$ are obtained by the algorithm EQUIVALENCE-CLASSES$(k,4)$. One can check that for the vectors

$$u = (00001010\infty1\infty\infty10\infty\infty\infty001\infty1)$$

$$v = (0000100\infty00\infty10001\infty111\infty)$$

there is a representative from each orbit 2-15 on 276 of the $d[x,y,z]$ classes. Table 2.1 shows nine classes which do not have representative from all the orbits:

Table 2.1: List of classes not having representative from all the orbits for $k = 21$ and $g = 3$

Class	Missing orbits
$d[1,2,2]$	10
$d[1,5,6]$	2
$d[1,6,12]$	5
$d[1,13,5]$	9
$d[2,3,8]$	6
$d[2,7,3]$	10
$d[2,12,3]$	13
$d[3,6,8]$	6
$d[3,7,7]$	10

In order to complete the covering conditions, we add a small matrix C_1.

$$C_1 = \begin{pmatrix}
\infty & \infty & 0 & 0 & 1 & \infty & \infty & 0 & 0 & 1 & \infty & \infty & 0 & 0 & 0 & \infty & \infty & 1 & 0 & 0 & \infty \\
0 & 1 & 1 & \infty & 0 & 0 & \infty & 1 & 0 & 0 & \infty & 1 & 1 & \infty & 0 & 0 & 1 & 1 & \infty & \infty & 0 \\
1 & 1 & \infty & 0 & 0 & 0 & \infty & \infty & 1 & 0 & \infty & \infty & 1 & 1 & 0 & 0 & 0 & 1 & \infty & \infty & 0 \\
1 & \infty & 1 & 0 & 0 & \infty & 1 & 1 & 0 & 0 & \infty & 1 & 0 & 0 & \infty & 1 & 0 & 1 & 0 & \infty & 1 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & \infty & 1 & 1 & 0 & 0 & \infty & \infty & 1 & \infty & 0 & 0 & 1 & \infty & \infty \\
\infty & 0 & 0 & 0 & \infty & \infty & 0 & 0 & 0 & \infty & 0 & 1 & 1 & \infty & \infty & 0 & 1 & 0 & 0 & \infty & 0 \\
\infty & 0 & 1 & 0 & 1 & \infty & 0 & 1 & 0 & 1 & 1 & \infty & \infty & 0 & 1 & 0 & 1 & 1 & 1 & 1 & \infty \\
\infty & 1 & \infty & 0 & \infty & \infty & 1 & 1 & 0 & \infty & \infty & \infty & 1 & 0 & 0 & 0 & 1 & 0 & \infty & 0 & \infty \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \infty & 0 & 0 & 0 & \infty & 0 & 0 & 1 & 0 & 1
\end{pmatrix}^T$$

We use computer search to find matrix C_1. This matrix has the property that every choice of four rows in $[1,2,2]$, $[2,7,3]$ and $[3,7,7]$ contains at least one representative from orbit 10; every choice of four rows in $[2,3,8]$ and $[3,6,8]$ contains at least one representative from orbit 6; each choice of four rows in $[1,5,6]$, $[1,6,12]$, $[1,13,5]$ and $[2,12,3]$ contains at least one representative from orbit 2, 5, 9 and 13 respectively. We also need to use the following matrix

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

to ensure the coverage of all identical 4-tuples. Therefore, $[M^G, C_1^G, C]$ is a 4-$CA(309,21,3)$.

2.3.4 Case 3: One vector u and a matrix C_1

For $k = 37$ to 58, it is enough to use one vector u and a C_1 matrix of order $k \times \ell$ with $\ell < k$. For vector $u = (u_0, u_1, \ldots, u_{k-1})$, we define the sets $d[x,y,z]$ for positive integers x,y,z as follows:

$$d[x,y,z] = \left\{(u_i, u_{i+x}, u_{i+x+y}, u_{i+x+y+z}) : 0 \le i \le k-1\right\}$$

where the subscripts are taken modulo k. Vector $u = (u_0, u_1, \dots, u_{k-1})$ is said to be a vector with good 4-way configuration coverage, if each $d[x,y,z]$ class has a representative from most of the orbits 2-15. We use the vector to form a $k \times k$ matrix

$$M = \begin{pmatrix} u_0 & u_{k-1} & \cdots & u_1 \\ u_1 & u_0 & \cdots & u_2 \\ \vdots & \vdots & & \vdots \\ u_{k-2} & u_{k-3} & \cdots & u_{k-1} \\ u_{k-1} & u_{k-2} & \cdots & u_0 \end{pmatrix}.$$

In order to complete the covering conditions, we add a small matrix C_1. Therefore, $[M^G, C_1^G, C]$ is a strength four covering array.

Example 2.3.3. Let $k = 39$ and $g = 3$. Here we are unable to find a vector u such that each $d[x,y,z]$ contains a representative from each of the orbits 2-15. For $k = 39$, there are 2109 $[x,y,z]$ classes. The vector

$$u = (001\infty\infty11\infty11\infty0001\infty11\infty101\infty\infty\infty1\infty0\infty0010\infty00\infty\infty0)$$

is a vector with high 4-way configuration coverage measure, that is each $d[x,y,z]$ class corresponds to u has a representative from most of the orbits 2-15. In order to complete the covering conditions, we use computer search to find a 39×34 matrix C_1. To ensure the coverage of all identical 4-tuples we concatenate the matrix C. Therefore, $[M^G,\ C_1^G,\ C]$ is a 4-$CA(441, 39, 3)$.

2.4 Improving the solutions

We examine two methods to obtain small improvements on the computational results obtained.

2.4.1 Extending a solution

Until this point, covering arrays have been developed by applying a cyclic rotation of the starter vectors in addition to the action of PGL on the symbols. As in [69], one can also consider fixing one row, and developing the remaining $k-1$ cyclically. This can be viewed as first finding a solution of the type already described on $k-1$ rows, but requiring an additional property. For the 4-subsets of $\{0,\ldots,k-2\}$, equivalence classes are defined as before, with arithmetic modulo $k-1$:

$$[\{s_1,s_2,s_3,s_4\}] = \big\{\{s_1+d,s_2+d,s_3+d,s_4+d\} \mid 0 \le d \le k-2\big\}.$$

For 3-subsets $\{t_1,t_2,t_3\}$ of $\{0,\ldots,k-2\}$ we define further equivalence classes as

$$[\{t_1,t_2,t_3,k-1\}] = \big\{\{t_1+d,t_2+d,t_3+d,k-1\} \mid 0 \le d \le k-2\big\}.$$

If we can place an entry in position $k-1$ to extend the length of each starter vector so that every one of the (old and new) equivalence classes represents each of the orbits 2-15, we obtain a strength four covering array of degree k. We show an example to explain the method.

Example 2.4.1. This example explains how a covering array of degree $k=35$ can be extended to one of degree $k=36$ without increasing the size of the covering array. For the 4-subsets of $\{0,\ldots,34\}$, there are total 1496 equivalence classes obtained from algorithm Equivalence-Classes$(k,4)$. Two length-35 starter vectors are shown below:

$$u = (01\infty0\infty\infty1000\infty01\infty\infty0\infty1\infty111\infty\infty\infty01\infty01000\infty1)$$
$$v = (0\infty00111\infty0\infty110\infty11\infty110\infty010010000\infty1\infty\infty0)$$

Each new equivalence class $[\{t_1,t_2,t_3,35\}]$ corresponds to a 3-subset $\{t_1,t_2,t_3\}$ of $\{0,\ldots,34\}$ represents each of the orbits 2-14 when $u_{35}=\infty$ and $v_{35}=\infty$. We form a 36×70 starter matrix M for $k=36$ as follows:

$$M_1 = \begin{pmatrix} u_0 & u_{k-1} & \dots & u_1 & v_0 & v_{k-1} & \dots & v_1 \\ u_1 & u_0 & \dots & u_2 & v_1 & v_0 & \dots & v_2 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ u_{k-2} & u_{k-3} & \dots & u_{k-1} & v_{k-2} & v_{k-3} & \dots & v_{k-1} \\ u_{k-1} & u_{k-2} & \dots & u_0 & v_{k-1} & v_{k-2} & \dots & v_0 \\ \infty & \infty & \dots & \infty & \infty & \infty & \dots & \infty \end{pmatrix}.$$

Now $[M^G,\ C]$ is a 4-$CA(423,36,3)$ obtained by extending 4-$CA(423,35,3)$.

The potential advantage of this approach is that a solution for degree $k-1$ can sometimes be extended to one of degree k without increasing the size of the covering array produced. Indeed we found that the solutions for $k-1 \in \{32,34,35\}$ do ensure that the new equivalence classes also represent each of the orbits 2-15. Hence we obtain the following improvements. Old indicates the bound obtained by applying our methods to k; Improved gives the bound by applying the method to $k-1$ and ensuring that the new equivalence classes represent all orbits:

Table 2.2: Improved bounds on 4-$CAN(k,3)$ obtained by extending a solution.

k	Old	Improved	k	Old	Improved	k	Old	Improved
33	399	387	35	423	411	36	435	423

2.4.2 Randomized Post-optimization

Nayeri, Colbourn, and Konjevod [73] describe a post-optimization strategy which, when applied to a covering array, exploits the flexibility of symbols in an attempt to reduce its size. We applied their method to the arrays provided here, and to arrays obtained by removing one or more rows. Because the method is described in detail in [73], we simply report improvements for eight values of k. Basic gives the bound from starter vectors, Improved gives the bound on 4-$CAN(k,3)$ after post-optimization:

Table 2.3: Improved bounds on 4-$CAN(k,3)$ obtained by randomized post-optimization.

k	Basic	Improved	k	Basic	Improved
19	309	300	20	309	303
21	309	305	22	309	307
27	351	345	28	363	360
34	411	410	37	435	433

2.5 Results

Tables 2.4, 2.5, 2.6 and 2.7 give a list of starter vectors and matrix C_1 that improves the best known bounds. The old bounds are from [30]. When the new bound is marked with an asterisk, post-optimization has been applied (see Section 2.4).

Table 2.4: Improved strength four covering arrays for $g = 3$.

k	Starter vectors and matrix C_1	New bound	Old bound [30]
21	$u = (00001010\infty1\infty\infty10\infty\infty001\infty1)$ $v = (0000100\infty00\infty10001\infty111\infty)$ $C_1 = \begin{pmatrix} \infty & \infty & 0 & 0 & 1 & \infty & \infty & 0 & 0 & 1 & \infty & \infty & 0 & 0 & 0 & \infty & \infty & 1 & 0 & 0 & \infty \\ 0 & 1 & 1 & \infty & 0 & 0 & \infty & 1 & 0 & 0 & \infty & 1 & 1 & \infty & 0 & 0 & 1 & 1 & \infty & \infty & 0 \\ 1 & 1 & \infty & 0 & 0 & 0 & \infty & \infty & 1 & 0 & \infty & \infty & 1 & 1 & 0 & 0 & 0 & 1 & \infty & \infty & 0 \\ 1 & \infty & 1 & 0 & 0 & \infty & 1 & 1 & 0 & 0 & \infty & 1 & 0 & 0 & \infty & 1 & 0 & 1 & 0 & \infty & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & \infty & 1 & 1 & 0 & 0 & \infty & \infty & 1 & \infty & 0 & 0 & 1 & \infty & \infty \\ \infty & 0 & 0 & 0 & \infty & \infty & 0 & 0 & 0 & \infty & 0 & 1 & 1 & \infty & \infty & 0 & 1 & 0 & 0 & \infty & 0 \\ \infty & 0 & 1 & 0 & 1 & \infty & 0 & 1 & 0 & 1 & 1 & \infty & \infty & 0 & 1 & 0 & 1 & 1 & 1 & 1 & \infty \\ \infty & 1 & \infty & 0 & \infty & \infty & 1 & 1 & 0 & \infty & \infty & \infty & 1 & 0 & 0 & 0 & 1 & 0 & \infty & 0 & \infty \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \infty & 0 & 0 & 0 & \infty & 0 & 0 & 1 & 0 & 1 \end{pmatrix}^T$	305*	315
22	$u = (0000011\infty0\infty0110\infty1\infty\infty\infty01\infty)$ $v = (00010010\infty1\infty\infty0\infty01\infty10\infty\infty1)$ $C_1 = \begin{pmatrix} 0 & \infty & \infty & 0 & 0 & 0 & \infty & \infty & \infty & 0 & 0 & 0 & \infty & \infty & 0 & 0 & 0 & \infty & \infty & \infty & 0 & 0 \\ \infty & \infty & 0 & 0 & 0 & \infty & \infty & \infty & 0 & 0 & \infty & \infty & \infty & 0 & 0 & 0 & \infty & \infty & \infty & 0 & 0 & \infty \\ 1 & \infty & 1 & \infty & 0 & \infty & 0 & 1 & \infty & 1 & \infty & 1 & \infty & 1 & \infty & 0 & 0 & 0 & 1 & \infty & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & \infty & 0 & \infty & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \infty & 1 & \infty \\ \infty & 0 & 0 & \infty & \infty & 1 & 0 & 1 & \infty & 0 & 0 & \infty & \infty & \infty & 0 & 0 & 1 & \infty & 0 & 0 & 1 & \infty \\ \infty & 0 & \infty & 1 & 1 & 1 & 0 & 1 & \infty & 1 & \infty & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & \infty & 0 \\ 0 & 0 & 0 & \infty & \infty & 1 & 0 & 0 & \infty & \infty & \infty & 0 & \infty & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & \infty \end{pmatrix}^T$	307*	315

Table 2.5: Improved strength four covering arrays for $g = 3$ (continued).

k	Starter vectors and matrix C_1	New bound	Old bound [30]
27	$u = (1101011\infty\infty\infty0\infty00\infty\infty1\infty011\infty0100\infty)$ $v = (11\infty0\infty1011\infty\infty\infty0\infty0\infty01\infty00001\infty\infty\infty)$ $C_1 = \begin{pmatrix} 0&1&0&1&0&1&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&1&0&1&0&1&0&1&0&1&0&1 \\ 0&\infty&0&\infty&0&\infty&0&\infty&0&\infty&0&\infty&0&\infty&0&\infty&0&\infty&0&\infty&0&\infty&0&\infty&\infty&0&\infty \\ 0&\infty&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&\infty&0&0 \end{pmatrix}^T$	345*	378
28	$u = (1\infty\infty00\infty\infty1\infty01101111\infty0\infty0101\infty\infty\infty1)$ $v = (\infty1011\infty110\infty000\infty1\infty\infty10\infty\infty0\infty00\infty01)$ $C_1 = \begin{pmatrix} \infty&0&\infty&0&0&\infty&0&0&\infty&0&\infty&\infty&0&\infty&\infty&0&\infty&0&0&\infty&0&0&\infty&0&\infty&\infty&0&\infty \\ \infty&0&0&1&0&1&\infty&0&\infty&1&0&\infty&\infty&0&1&0&0&\infty&0&\infty&1&0&1&\infty&0&1&1&0 \\ 1&0&\infty&0&\infty&\infty&0&\infty&\infty&1&\infty&1&0&0&0&1&\infty&1&0&\infty&1&\infty&1&0&\infty&0&1&1 \\ 0&\infty&0&\infty&0&0&0&0&\infty&0&1&0&1&0&\infty&0&1&0&\infty&0&0&1&0&0&0&0&0&0 \end{pmatrix}^T$	360*	383
30	$u = (011\infty11\infty\infty\infty001\infty\infty\infty1\infty10\infty\infty0\infty1100\infty01)$ $v = (11\infty\infty01101000\infty101\infty1\infty0\infty000010\infty\infty\infty)$	363	393
32	$u = (\infty1100010\infty111\infty1\infty010\infty\infty0100\infty\infty0\infty\infty010)$ $v = (\infty000\infty1\infty\infty0\infty000110\infty\infty100\infty0\infty11\infty11111)$	387	409
33	Obtained from $CA(387, 32, 3)$	387	417
34	$u = (00\infty101\infty\infty\infty1001\infty010\infty\infty0\infty0\infty01\infty\infty0\infty11111)$ $v = (1100\infty1\infty01\infty10110\infty\infty0\infty\infty011\infty101001\infty000)$	410*	423
35	Obtained from $CA(411, 34, 3)$	411	429

Table 2.6: Improved strength four covering arrays for $g = 3$ (continued).

k	Starter vectors and matrix C_1	New bound	Old bound [30]
35	$u = (01\infty0\infty\infty1000\infty01\infty\infty0\infty1\infty111\infty\infty\infty01\infty01000\infty1)$ $v = (0\infty00111\infty0\infty110\infty11\infty110\infty010010000\infty1\infty\infty0)$	423	429
36	Obtained from $CA(423, 35, 3)$	423	441
37	$u = (001\infty10\infty1\infty01000\infty1100\infty101111\infty001\infty\infty\infty\infty00\infty)$ C_1: 37×35 matrix	433*	441
39	$u = (001\infty\infty11\infty11\infty0001\infty11\infty101\infty\infty\infty1\infty0\infty0010\infty00\infty\infty0)$ C_1: 39×34 matrix	441	453
41	$u = (\infty001\infty010\infty\infty0\infty0101111\infty\infty011\infty\infty10000\infty0\infty\infty10\infty0\infty1)$ C_1: 41×34 matrix	453	465
42	$u = (\infty0111\infty1\infty\infty100\infty101\infty01000\infty011\infty1010011\infty00\infty1\infty\infty\infty)$ C_1: 42×35 matrix	465	471
46	$u = (\infty00000\infty1100010\infty101\infty\infty1\infty01\infty00110\infty\infty\infty\infty11\infty1101\infty101\infty)$ C_1: 46×33 matrix	477	483
47	$u = (\infty0011\infty1101\infty1\infty000\infty1\infty01\infty00\infty111010\infty00\infty\infty\infty10\infty\infty1\infty\infty1\infty\infty)$ C_1: 47×33 matrix	483	489
48	$u = (01\infty\infty\infty11\infty01\infty1010111\infty\infty001\infty\infty\infty0\infty110010\infty0\infty\infty000100\infty00\infty)$ C_1: 48×33 matrix	489	495
51	$u = (\infty0\infty\infty101011\infty000\infty\infty11\infty1\infty1001\infty\infty\infty\infty11$ $\infty0\infty1\infty01111001001\infty00)$ C_1: 51×32 matrix	501	507

Table 2.7: Improved strength four covering arrays for $g = 3$ (continued).

k	Starter vectors and matrix C_1	New bound	Old bound [30]
55	$u = (1\infty\infty1\infty1\infty0\infty111\infty\infty1\infty0010\infty00\infty0011011\infty1\infty0$ $00\infty11\infty\infty0101\infty001110\infty\infty)$ C_1: 55×30 matrix	513	519
57	$u = (\infty10\infty\infty\infty0011\infty01\infty10\infty11001\infty1\infty\infty0011\infty\infty110$ $110111010\infty\infty1\infty0\infty0000\infty01)$ C_1: 57×29 matrix	519	531
58	$u = (\infty0\infty\infty00101\infty0010\infty0\infty1\infty1000\infty0\infty11001\infty00010\infty111$ $\infty\infty\infty11011011\infty\infty0\infty0\infty)$ C_1: 58×29 matrix	525	531
63	$u = (1101\infty10\infty100\infty\infty\infty00101\infty\infty0\infty0\infty\infty1\infty010\infty11\infty\infty\infty01$ $10\infty10110001\infty0\infty11\infty\infty0\infty0\infty11)$ C_1: 63×26	537	549
67	$u = (010101\infty1100\infty100\infty11\infty\infty\infty\infty0110\infty01111\infty\infty1011\infty0\infty$ $1101\infty0\infty\infty0\infty101\infty\infty1\infty\infty10000\infty00)$ C_1: 67×25	555	561
70	$u = (1\infty001\infty11\infty1\infty\infty\infty0\infty11\infty0\infty0\infty1\infty00011\infty0\infty\infty\infty\infty111$ $\infty0101001\infty010011\infty\infty010000\infty10\infty\infty1100)$ C_1: 70×24	567	573
72	$u = (\infty\infty000\infty1010\infty\infty\infty\infty\infty010111000\infty11011\infty011101\infty0\infty\infty1\infty00$ $\infty1\infty1\infty\infty010\infty101100\infty01\infty\infty\infty1\infty\infty0\infty)$ C_1: 72×24	573	579
74	$u = (1\infty0010\infty\infty01\infty\infty\infty111\infty\infty1\infty\infty0100\infty\infty\infty\infty10\infty1011011\infty$ $001100001\infty\infty0\infty0\infty0\infty\infty101100\infty1\infty01\infty111\infty)$ C_1: 74×24	585	591

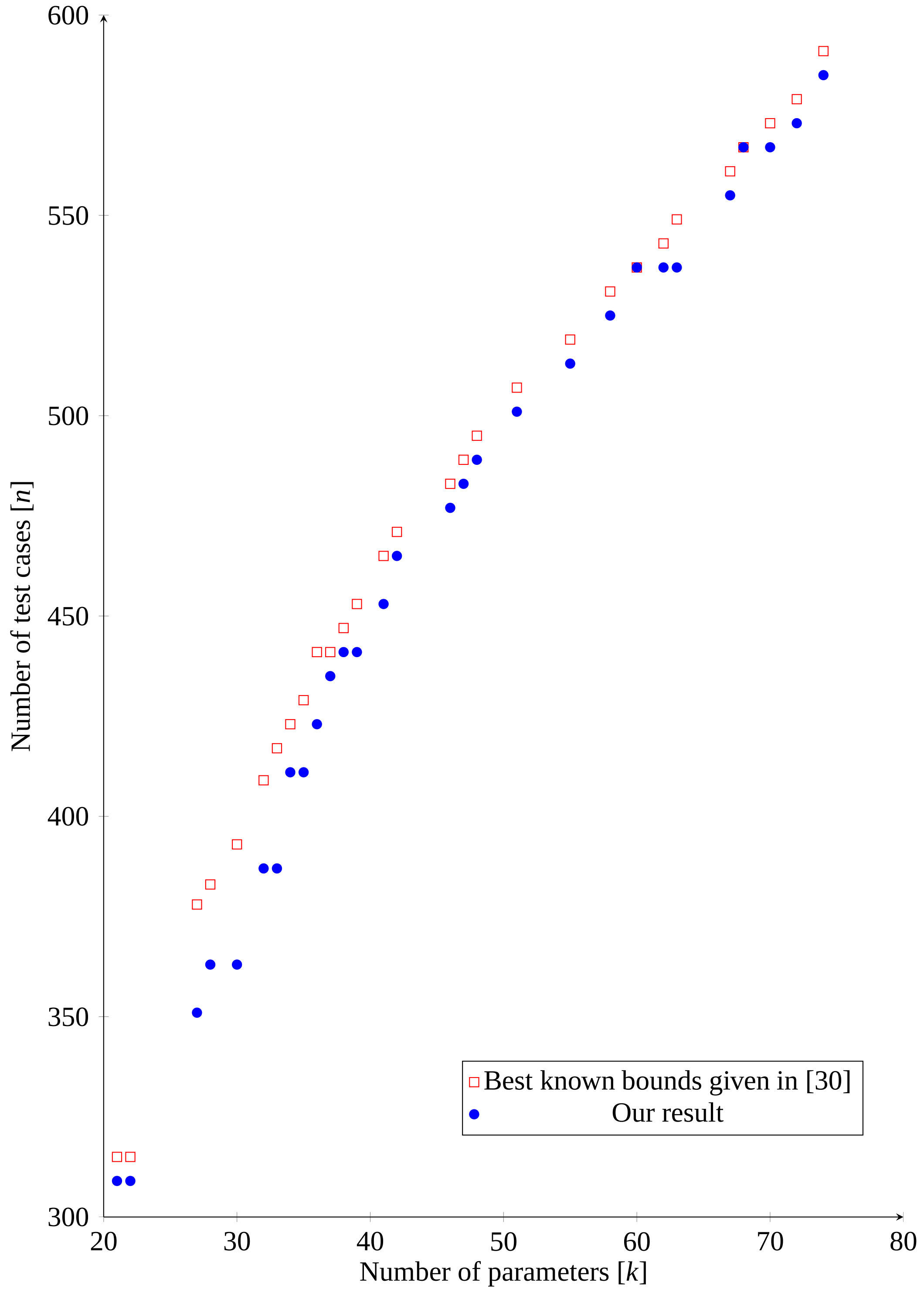


Figure 2.1: Scatter plot for improved bounds on 4-$CAN(k,3)$

Chapter 3

Testing Arrays with High Coverage Measure

In most software development environments, time, computing and human resources needed to perform the testing of a component is strictly limited. Given the different input parameters with multiple possible values for each parameter, performing exhaustive testing which tests all possible test cases is practically impossible. When testing a software system with k parameters, each of which must be tested with g values, the total number of possible test cases is g^k. For instance, if there are 20 parameters and three values for each parameter then the number of input combinations or test cases of this system is $3^{20} = 3486784401$. A fundamental problem with software testing is that testing under all combinations of inputs is not feasible, even with a simple product [53, 58]. Budgets assigned for software testings are generally limited. Software developers cannot test everything, but they can use combinatorial test design to identify a small number of test cases with high configuration coverage. The goal of the most combinatorial testing research is to create test suites that find a large percentage of errors of a system while having a small number of tests required. To model this situation, we consider the problem of determining a testing array with high configuration coverage measure within a fixed number of test cases. More formally, given fixed values of t,k,g and n our objective is to build a testing array $\mathscr{A}$ of size at most n having high t-way configuration coverage measure. This chapter presents algebraic constructions for testing arrays with high 3- and 4-way configuration coverage measure. See also [7, 6]. In Section 3.1, we recall the definitions of combinatorial coverage and configuration coverage from [48, 56]. In Section 3.2, we present an algebraic construction for testing arrays with high 3-way configuration coverage measure. In Section 3.3, we present another algebraic construction for testing arrays with high 4-way configuration coverage measure. In Section 3.4, we present the computational results.

3.1 Preliminary

Let n, k and g be positive integers. A *testing array* $\mathscr{A}$ is a $k \times n$ array in which entries are from a finite set of g symbols. Each row of the testing array corresponds to a parameter, each column corresponds to a test case, and the g symbols correspond to the values for each parameter.

Example 3.1.1. Here is an example of a testing array $\mathscr{A}$ for a system with four parameters $A, B, C,$ and D each of which having two values or symbols.

$$\begin{array}{|c|cccccccc|}
\hline
A & a_0 & a_1 & a_1 & a_0 & a_1 & a_0 & a_0 & a_1 \\
B & b_0 & b_1 & b_1 & b_0 & b_0 & b_1 & b_1 & b_0 \\
C & c_0 & c_1 & c_0 & c_1 & c_0 & c_0 & c_1 & c_1 \\
D & d_0 & d_1 & d_0 & d_1 & d_1 & d_1 & d_0 & d_1 \\
\hline
\end{array}$$

Definition 3.1.1. For a set of t parameters, a *t-way parameter-value configuration* is an ordered tuple of t valid values, one for each of the parameters.

For $t = 3$, (b_1, c_0, d_0) and (b_1, c_1, d_0) are two different parameter-value configurations for parameters $B, C,$ and D. We now recall two types of coverage measure from [48, 56] and refer these articles for motivation.

Definition 3.1.2. For a given set of k parameters, *simple t-way combination coverage* is the proportion of t-way combinations of k parameters for which all parameter-value configurations are fully covered.

In Example 3.1.1, out of four 3-way combinations *ABC*, *ABD*, *ACD*, and *BCD*, only *ABC* has all eight 3-way parameter-value configurations covered, so the simple 3-way combination coverage for the test array $\mathscr{A}$ is $\frac{1}{4} = 25\%$.

When t parameters with g values each are considered, there are g^t t-way parameter-value configurations. So for k parameters with g values each, there are $\binom{k}{t}g^t$ possible t-way parameter-value configurations to be covered in a strength t covering array. We now recall a measure with respect to the number of parameter-value configurations covered.

Definition 3.1.3. The *t-way configuration coverage* $\mu_t(\mathscr{A})$ of a testing array $\mathscr{A}$ is defined by the ratio between the number of t-way parameter-value configurations contained in the column vectors of $\mathscr{A}$ and the total number of t-way parameter-value configurations given by $\binom{k}{t}g^t$.

If the testing array is a covering array, then both the simple combination coverage and configuration coverage measure are 100%. In Example 3.1.1, there are $\binom{4}{3} = 4$ possible parameter combinations and $\binom{4}{3}2^3 = 32$ possible 3-way parameter-value configurations. Among these 32 3-way parameter-value configurations, 29 3-way parameter-value configurations are covered and missing ones are (b_0, c_1, d_0), (a_1, b_0, d_0), and (a_1, c_1, d_0). Thus we have $\mu_3(\mathscr{A}) = \frac{29}{32} = 90.6\%$ for 3-way configuration coverage measure. Our objective is to construct a testing array $\mathscr{A}$ of size at most n having high t-way configuration coverage measure, given fixed values of t, k, g and n. This problem is also called *covering arrays with budget constraints*.

3.2 Construction of testing arrays with high $\mu_3(\mathscr{A})$

Given fixed values of k, n and g, so that $g-2$ is a prime power, we are to construct a testing array $\mathscr{A}$ with high 3-way configuration coverage measure. Let q be a prime power and F_q be the finite field with q elements. Let $X = \{F_q, \infty_1, \infty_2\}$ be the set of g symbols (values) on which we are to construct a testing array having good 3-way configuration coverage measure. Clearly, $|X| = g = q+2$; we choose g so that $g-2$ is a prime or prime power. Our construction requires selecting a group G and finding a vector $v \in X^k$, called a vector with good configuration coverage. We use the vector $v = (v_0, v_1, \dots, v_{k-1})$ to form a $k \times k$ circulant matrix

$$M = \begin{pmatrix} v_0 & v_{k-1} & \dots & v_1 \\ v_1 & v_0 & \dots & v_2 \\ \vdots & \vdots & & \vdots \\ v_{k-1} & v_{k-2} & \dots & v_0 \end{pmatrix}.$$

Let $G = AGL(1,q) = \left\{ \alpha \ : \ F_q \to F_q \mid x\alpha = ax+b; a \neq 0 \text{ and } a,b \in F_q \right\}$ be the set of all linear transformations. Note that G is a group with respect to functional composition and $|G| = q(q-1)$. By Theorem 2.1.2, the group is sharply 2-transitive on F_q. Consider the stabilizer $\text{stab}_G(x)$ for each $x \in F_q$. If $\text{stab}_G(x)$ is nontrivial for each $x \in F_q$, we define H to be a non-empty subset of G such that

1. $H = q$ and H does not contain the identity transformation of G
2. $|H \cap \text{stab}_G(x)| = 1$ for every $x \in F_q$.

If $\text{stab}_G(x)$ is trivial for some x, then H is empty. We define an action of G on the set $\{\infty_1, \infty_2\}$ as follows:

$$\infty_i \alpha = \begin{cases} \infty_j, & \text{if } \alpha \in H \text{ where } i \neq j \\ \infty_i, & \text{otherwise} \end{cases}$$

If H is non-empty then the action is transitive on $\{\infty_1, \infty_2\}$. This action together with natural action of G on F_q exhibits an action of G on X. This action of G on 3-tuples from X has the following orbits:

(1) $\{(a,a,a)^T : a \in F_q\}$

(2) $\{(\infty_1, \infty_1, \infty_1)^T, (\infty_2, \infty_2, \infty_2)^T\}$

(3) $\{(\infty_1,\infty_1,\infty_2)^T,(\infty_2,\infty_2,\infty_1)^T\}$

(4) $\{(\infty_1,\infty_2,\infty_1)^T,(\infty_2,\infty_1,\infty_2)^T\}$

(5) $\{(\infty_2,\infty_1,\infty_1)^T,(\infty_1,\infty_2,\infty_2)^T\}$

(6) $\{(a,a,\infty_i)^T : a \in F_q\}$

(7) $\{(a,\infty_i,a)^T : a \in F_q\}$

(8) $\{(\infty_i,a,a)^T : a \in F_q\}$

(9) $\{(a,\infty_i,\infty_i)^T : a \in F_q\}$

(10) $\{(\infty_i,a,\infty_i)^T : a \in F_q\}$

(11) $\{(\infty_i,\infty_i,a)^T : a \in F_q\}$

(12) $\{(a,\infty_i,\infty_j)^T : a \in F_q, i \neq j\}$

(13) $\{(\infty_i,a,\infty_j)^T : a \in F_q, i \neq j\}$

(14) $\{(\infty_i,\infty_j,a)^T : a \in F_q, i \neq j\}$

(15) $\{(a,b,\infty_i)^T : a,b \in F_q, a \neq b\}$

(16) $\{(a,\infty_i,b)^T : a,b \in F_q, a \neq b\}$

(17) $\{(\infty_i,a,b)^T : a,b \in F_q, a \neq b\}$

(18) $\{(a,a,b)^T : a,b \in F_q, a \neq b\}$

(19) $\{(a,b,a)^T : a,b \in F_q, a \neq b\}$

(20) $\{(b,a,a)^T : a,b \in F_q, a \neq b\}$

(21) $q-2$ orbits of the form $\{(a,b,c)^T : a,b,c \in F_q, a \neq b \neq c\}$.

Depending on the value of g there could be more than one orbit of the form 6-17. For v to be a starter vector, any three rows in the matrix M must have at least one representative from each of the orbits 6-21. If starter vector is not found, we look for a vector v with good configuration coverage measure. Vector $v = (v_0, v_1, \ldots, v_{k-1})$ is said to be a vector with good 3-way configuration coverage, if every $3 \times k$ subarray of M has a representative from most of the orbits 6-21. We also need to use $C_1 = 3\text{-}CA(n', k, 2)$, a minimum size covering array with entries from $\{\infty_1, \infty_2\}$ to ensure the coverage of all triples in the orbits 2-5. Let C be the $k \times q$ matrix that has a constant column with each entry equal to x, for each $x \in F_q$. We use C to ensure the coverage of all triples in orbit 1. The group $AGL(1, g-2)$ acting on the matrix M produces several matrices that are concatenated with C_1 and C to form a testing array of size $k(g-2)(g-3) + n' + g - 2$ having good 3-way configuration coverage. More formally, if $\alpha \in AGL(1, g-2)$, then M^{α} is the $k \times k$ matrix where the $[i, j]$th entry is $M[i, j]^{\alpha}$, the image of $M[i, j]$ under α. The matrix obtained by developing M by $AGL(1, g-2)$ is the $k \times k(g-2)(g-3)$ matrix

$$M^{AGL(1,g-2)} = [M^{\alpha} \ : \ \alpha \in AGL(1, g-2)].$$

Thus, if $k(g-2)(g-3) + n' + g - 2 \leq n$, then $\left[M^{AGL(1,g-2)}, C_1, C\right]$ is a testing array of size less than or equal to n having high 3-way configuration coverage. We give two examples to illustrate the method.

Example 3.2.1. Let $g = 5$, $k = 32$, $X = GF(3) \cup \{\infty_1, \infty_2\}$ and $G = AGL(1, 3)$. The elements of G are $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$ and α_6 where

$$x\alpha_1 = x \qquad x\alpha_2 = x + 1 \qquad x\alpha_3 = x + 2$$
$$x\alpha_4 = 2x \qquad x\alpha_5 = 2x + 1 \qquad x\alpha_6 = 2x + 2$$

for all $x \in GF(3)$. The stabilizers of elements from $GF(3)$ are $\text{stab}_G(0) = \{\alpha_1, \alpha_4\}$, $\text{stab}_G(1) = \{\alpha_1, \alpha_6\}$ and $\text{stab}_G(2) = \{\alpha_1, \alpha_5\}$. As α_1 is the identity of $AGL(1, 3)$, we set $H = \{\alpha_4, \alpha_5, \alpha_6\}$. Thus the action of G on $\{\infty_1, \infty_2\}$ is given by

$$\infty_1\alpha_1 = \infty_1, \infty_1\alpha_2 = \infty_1, \infty_1\alpha_3 = \infty_1, \infty_1\alpha_4 = \infty_2, \infty_1\alpha_5 = \infty_2, \infty_1\alpha_6 = \infty_2$$
$$\infty_2\alpha_1 = \infty_2, \infty_2\alpha_2 = \infty_2, \infty_2\alpha_3 = \infty_2, \infty_2\alpha_4 = \infty_1, \infty_2\alpha_5 = \infty_1, \infty_2\alpha_6 = \infty_1.$$

The action of G on 3-tuples from X has 24 orbits:

Orb 1: $\{(0,0,0)^T,(1,1,1)^T,(2,2,2)^T\}$

Orb 2: $\{(\infty_1,\infty_1,\infty_1)^T,(\infty_2,\infty_2,\infty_2)^T\}$

Orb 3: $\{(\infty_1,\infty_1,\infty_2)^T,(\infty_2,\infty_2,\infty_1)^T\}$

Orb 4: $\{(\infty_1,\infty_2,\infty_1)^T,(\infty_2,\infty_1,\infty_2)^T\}$

Orb 5: $\{(\infty_1,\infty_2,\infty_2)^T,(\infty_2,\infty_1,\infty_1)^T\}$

Orb 6: $\{(0,0,\infty_1)^T,(1,1,\infty_1)^T,(2,2,\infty_1)^T,(0,0,\infty_2)^T,(1,1,\infty_2)^T,(2,2,\infty_2)^T\}$

Orb 7: $\{(0,\infty_1,0)^T,(1,\infty_1,1)^T,(2,\infty_1,2)^T,(0,\infty_2,0)^T,(1,\infty_2,1)^T,(2,\infty_2,2)^T\}$

Orb 8: $\{(\infty_1,0,0)^T,(\infty_1,1,1)^T,(\infty_1,2,2)^T,(\infty_2,0,0)^T,(\infty_2,1,1)^T,(\infty_2,2,2)^T\}$

Orb 9: $\{(0,\infty_1,\infty_1)^T,(1,\infty_1,\infty_1)^T,(2,\infty_1,\infty_1)^T,(0,\infty_2,\infty_2)^T,(1,\infty_2,\infty_2)^T,(2,\infty_2,\infty_2)^T\}$

Orb 10: $\{(\infty_1,0,\infty_1)^T,(\infty_1,1,\infty_1)^T,(\infty_1,2,\infty_1)^T,(\infty_2,0,\infty_2)^T,(\infty_2,1,\infty_2)^T,(\infty_2,2,\infty_2)^T\}$

Orb 11: $\{(\infty_1,\infty_1,0)^T,(\infty_1,\infty_1,1)^T,(\infty_1,\infty_1,2)^T,(\infty_2,\infty_2,0)^T,(\infty_2,\infty_2,1)^T,(\infty_2,\infty_2,2)^T\}$

Orb 12: $\{(0,\infty_1,\infty_2)^T,(1,\infty_1,\infty_2)^T,(2,\infty_1,\infty_2)^T,(0,\infty_2,\infty_1)^T,(1,\infty_2,\infty_1)^T,(2,\infty_2,\infty_1)^T\}$

Orb 13: $\{(\infty_1,0,\infty_2)^T,(\infty_1,1,\infty_2)^T,(\infty_1,2,\infty_2)^T,(\infty_2,0,\infty_1)^T,(\infty_2,1,\infty_1)^T,(\infty_2,2,\infty_1)^T\}$

Orb 14: $\{(\infty_1,\infty_2,0)^T,(\infty_1,\infty_2,1)^T,(\infty_1,\infty_2,2)^T,(\infty_2,\infty_1,0)^T,(\infty_2,\infty_1,1)^T,(\infty_2,\infty_1,2)^T\}$

Orb 15 I: $\{(0,1,\infty_1)^T,(1,2,\infty_1)^T,(2,0,\infty_1)^T,(0,2,\infty_2)^T,(1,0,\infty_2)^T,(2,1,\infty_2)^T\}$

Orb 15 II: $\{(0,1,\infty_2)^T,(1,2,\infty_2)^T,(2,0,\infty_2)^T,(0,2,\infty_1)^T,(1,0,\infty_1)^T,(2,1,\infty_1)^T\}$

Orb 16 I: $\{(0,\infty_1,1)^T,(1,\infty_1,2)^T,(2,\infty_1,0)^T,(0,\infty_2,2)^T,(1,\infty_2,0)^T,(2,\infty_2,1)^T\}$

Orb 16 II: $\{(0,\infty_2,1)^T,(1,\infty_2,2)^T,(2,\infty_2,0)^T,(0,\infty_1,2)^T,(1,\infty_1,0)^T,(2,\infty_1,1)^T\}$

Orb 17 I: $\{(\infty_1,0,1)^T,(\infty_1,1,2)^T,(\infty_1,2,0)^T,(\infty_2,0,2)^T,(\infty_2,1,0)^T,(\infty_2,2,1)^T\}$

Orb 17 II: $\{(\infty_2,0,1)^T,(\infty_2,1,2)^T,(\infty_2,2,0)^T,(\infty_1,0,2)^T,(\infty_1,1,0)^T,(\infty_1,2,1)^T\}$

Orb 18: $\{(0,0,1)^T,(1,1,2)^T,(2,2,0)^T,(0,0,2)^T,(1,1,0)^T,(2,2,1)^T\}$

Orb 19: $\{(0,1,0)^T,(1,2,1)^T,(2,0,2)^T,(0,2,0)^T,(1,0,1)^T,(2,1,2)^T\}$

Orb 20: $\{(0,1,1)^T,(1,2,2)^T,(2,0,0)^T,(0,2,2)^T,(1,0,0)^T,(2,1,1)^T\}$

Orb 21: $\{(0,1,2)^T,(1,2,0)^T,(2,0,1)^T,(0,2,1)^T,(1,0,2)^T,(2,1,0)^T\}$

We use computer search to find a vector v with good 3-way configuration coverage measure, that is, each $3\times k$ sub-matrix of M has a representative from most of the orbits 6-21:

$$v = (\infty_2\infty_1 220\infty_1 210\infty_1 1\infty_1 2\infty_1 110111\infty_2 0002\infty_2 1\infty_1\infty_1 2\infty_1\infty_2).$$

Build the 32×32 matrix M from v. The action of $G=\{x,x+1,x+2,2x,2x+1,2x+2\}$ on M produces six matrices which are concatenated to get M^G. A small covering array $C_1 = 3\text{-}CA(24,32,2)$ on symbols $\{\infty_1,\infty_2\}$ and a 32×3 array C as shown below needs to be concatenated to cover interactions in orbits 1-5:

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

The matrix $\mathscr{A} = [M^G, C_1, C]$ is a 32×219 testing array with 3-way configuration coverage measure $\mu_3(\mathscr{A}) = 0.905$.

Example 3.2.2. Let $g=4$, $k=50$, $X=GF(2)\cup\{\infty_1,\infty_2\}$ and $G=AGL(1,2)$. The elements of G are α_1,α_2 where

$$x\alpha_1 = x \quad \text{and} \quad x\alpha_2 = x+1$$

for all $x\in GF(2)$. The stabilizers of elements from $GF(2)$ are $\text{stab}_G(0)=\{\alpha_1\}$ and $\text{stab}_G(1)=\{\alpha_1\}$. As α_1 is the identity of $AGL(1,2)$, we set $H=\varnothing$. Thus the action of G on $\{\infty_1,\infty_2\}$ is given by

$$\infty_1\alpha_1 = \infty_1, \infty_1\alpha_2 = \infty_1, \infty_2\alpha_1 = \infty_2, \infty_2\alpha_2 = \infty_2.$$

The action of G on 3-tuples from X has 36 orbits:

Orb 1: $\{(0,0,0)^T, (1,1,1)^T\}$

Orb 2 I: $\{(\infty_1,\infty_1,\infty_1)^T\}$

Orb 2 II: $\{(\infty_2,\infty_2,\infty_2)^T\}$

Orb 3 I: $\{(\infty_1,\infty_1,\infty_2)^T\}$

Orb 3 II: $\{(\infty_2,\infty_2,\infty_1)^T\}$

Orb 4 I: $\{(\infty_1,\infty_2,\infty_1)^T\}$

Orb 4 II: $\{(\infty_2,\infty_1,\infty_2)^T\}$

Orb 5 I: $\{(\infty_1,\infty_2,\infty_2)^T\}$

Orb 5 II: $\{(\infty_2,\infty_1,\infty_1)^T\}$

Orb 6 I: $\{(0,0,\infty_1)^T, (1,1,\infty_1)^T\}$

Orb 6 II: $\{(0,0,\infty_2)^T, (1,1,\infty_2)^T\}$

Orb 7 I: $\{(0,\infty_1,0)^T, (1,\infty_1,1)^T\}$

Orb 7 II: $\{(0,\infty_2,0)^T, (1,\infty_2,1)^T\}$

Orb 8 I: $\{(\infty_1,0,0)^T, (\infty_1,1,1)^T\}$

Orb 8 II: $\{(\infty_2,0,0)^T, (\infty_2,1,1)^T\}$

Orb 9 I: $\{(0,\infty_1,\infty_1)^T, (1,\infty_1,\infty_1)^T\}$

Orb 9 II: $\{(0,\infty_2,\infty_2)^T, (1,\infty_2,\infty_2)^T\}$

Orb 10 I: $\{(\infty_1,0,\infty_1)^T, (\infty_1,1,\infty_1)^T\}$

Orb 10 II: $\{(\infty_2,0,\infty_2)^T, (\infty_2,1,\infty_2)^T\}$

Orb 11 I: $\{(\infty_1,\infty_1,0)^T, (\infty_1,\infty_1,1)^T\}$

Orb 11 II: $\{(\infty_2,\infty_2,0)^T, (\infty_2,\infty_2,1)^T\}$

Orb 12 I: $\{(0,\infty_1,\infty_2)^T, (1,\infty_1,\infty_2)^T\}$

Orb 12 II: $\{(0,\infty_2,\infty_1)^T, (1,\infty_2,\infty_1)^T\}$

Orb 13 I: $\{(\infty_1,0,\infty_2)^T, (\infty_1,1,\infty_2)^T\}$

Orb 13 II: $\{(\infty_2,0,\infty_1)^T, (\infty_2,1,\infty_1)^T\}$

Orb 14 I: $\{(\infty_1,\infty_2,0)^T, (\infty_1,\infty_2,1)^T\}$

Orb 14 II: $\{(\infty_2,\infty_1,0)^T, (\infty_2,\infty_1,1)^T\}$

Orb 15 I: $\{(0,1,\infty_1)^T, (1,0,\infty_1)^T\}$

Orb 15 II: $\{(0,1,\infty_2)^T, (1,0,\infty_2)^T\}$

Orb 16 I: $\{(0,\infty_1,1)^T, (1,\infty_1,0)^T\}$

Orb 16 II: $\{(0,\infty_2,1)^T, (1,\infty_2,0)^T\}$

Orb 17 I: $\{(\infty_1,0,1)^T, (\infty_1,1,0)^T\}$

Orb 17 II: $\{(\infty_2,0,1)^T, (\infty_2,1,0)^T\}$

Orb 18: $\{(0,0,1)^T, (1,1,0)^T\}$

Orb 19: $\{(0,1,0)^T, (1,0,1)^T\}$

Orb 20: $\{(0,1,1)^T, (1,0,0)^T\}$

We use computer search to find a vector v with good 3-way configuration coverage measure, that is, each $3 \times k$ sub-matrix of M has a representative from most of the orbits 6 I-20:

$$v = (\infty_1 11\infty_2\infty_2\infty_2 0\infty_1 011011\infty_1\infty_1\infty_2 1\infty_2 1\infty_2\infty_2 0\infty_1\infty_2 1000\infty_2 001\infty_1 10\infty_2\infty_1 1101\infty_1\infty_2\infty_1\infty_1 0\infty_1\infty_1 1).$$

Build the 50×50 matrix M from v. The action of $G = \{x, x+1\}$ on M produces two matrices which are concatenated to get M^G. A small covering array $C_1 = 3\text{-}CA(28, 50, 2)$ on symbols $\{\infty_1, \infty_2\}$ and a 50×2 array C as shown below needs to be concatenated to cover interactions in orbits 1-5 II:

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

The matrix $\mathscr{A} = [M^G, C_1, C]$ is a 50×130 testing array with 3-way configuration coverage measure $\mu_3(\mathscr{A}) = 0.90202$.

3.2.1 Choice of vector v

The problem is to find a vector $v \in X^k$ with good 3-way configuration coverage measure, that is, each $3 \times k$ submatrix of M has a representative from most of the orbits 6-21. To determine which vector works as a vector with good 3-way configuration coverage, we define the sets $d[x,y]$ for positive integers x and y as follows:

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

where the subscripts are taken modulo k. For computational convenience, we partition the collection of $\binom{k}{3}$ choices of three distinct rows from k rows into disjoint equivalence classes. Formally, let S be the set of all $\binom{k}{3}$ 3-combinations of the set $\{0, 1, \ldots, k-1\}$. Define a binary relation R on S by putting

$$\{s_1, s_2, s_3\}\ R\ \{s'_1, s'_2, s'_3\} \text{ iff}$$

$$\{s_1 + d, s_2 + d, s_3 + d\} = \{s'_1, s'_2, s'_3\} \text{ for some } d \in \mathbb{N}$$

where the addition is modulo k. Because R is an equivalence relation on S, S can be partitioned into disjoint equivalence classes. The equivalence class determined by $\{s_1, s_2, s_3\} \in S$ is given by

$$[\{s_1, s_2, s_3\}] = \Big\{\{s_1 + d, s_2 + d, s_3 + d\} \mid 0 \le d \le k-1\Big\}.$$

Without loss of generality, we may assume that $0 = s_1 < s_2 < s_3$ for each equivalence class representative $[\{s_1, s_2, s_3\}]$. As an illustration, when $k = 6$, S is partitioned into four disjoint equivalence classes:

$$[\{0,1,2\}] \quad [\{0,1,3\}] \quad [\{0,1,4\}] \quad [\{0,2,4\}]$$

A distance vector (x,y,z) is associated with every equivalence class $[\{s_1, s_2, s_3\}]$ where $x = s_2 - s_1$, $y = s_3 - s_2$, $z = s_1 - s_3 \bmod k$. The third distance is redundant because $x + y + z = k$. We rewrite the equivalence class of 3-combinations $[\{s_1, s_2, s_3\}]$ as

$$[x,y] = \Big\{\{i, i+x, i+x+y\} \mid i = 0,1,2,...,k-1\Big\}.$$

For $k = 6$, $[1,1] = [\{0,1,2\}]$, $[1,2] = [\{0,1,3\}]$, $[1,3] = [\{0,1,4\}]$ and $[2,2] = [\{0,2,4\}]$.

Lemma 3.2.1. *Let S be the set of all 3-combinations of $\{0,1,2,...,k-1\}$. Then S can be partitioned into disjoint equivalence classes*

$$[x,y] = \Big\{\{i, i+x, i+x+y\} \mid i = 0,1,2,...,k-1\Big\}$$

where $x = 1,2,..., \lfloor \frac{k}{3} \rfloor$ and $y = x, x+1, ..., k-1$ such that

(i) $2x + y < k$

(ii) when $k \equiv 0 \ (mod\, 3)$, apart from above mentioned classes consider one more class $[\frac{k}{3}, \frac{k}{3}]$, that is, $x = y = z = \frac{k}{3}$.

There are no further classes distinct from these.

Before proving the result, we give an example. When S is the set of all 3-combinations of $\{0,1,2,3,4,5\}$, S can be partitioned into four disjoint classes: $[1,1]$, $[1,2]$, $[1,3]$ and $[2,2]$.

Proof. Let (x,y,z) be the distance vector corresponding to equivalence class $[\{s_1, s_2, s_3\}]$. Classes $[\{s_1, s_2, s_3\}]$, $[x,y]$, $[y,z]$ and $[z,x]$ are the same. Without loss of generality, we choose $[x,y]$ as class representative if $x \le y$ and $x \le z$. Thus $1 \le x \le \frac{k}{3}$ and $y = x, x+$

$1,\ldots,k-1$. It is enough to consider $z > x$. If $z = x$, then the classes $[x,y]$ and $[x,x]$ obtained from distance vector (x,y,x) are the same equivalence class. The classes of the form $[x,x]$ are generated when $y = x$. In order to avoid repetition, z has to be strictly greater than x. That is, $z = k-x-y > x$ which implies $2x+y < k$. When $k \equiv 0 \pmod 3$, the class $[\frac{k}{3},\frac{k}{3}]$ is not considered under the inequality $2x+y<k$. To include this class, we consider $x = y = z = \frac{k}{3}$ as well. Hence the lemma follows. $\square$

We present the following algorithm that generates all the equivalence classes without repetition.

Algorithm 2 Equivalence-Classes(k, 3)

```
Input: $k$
Output: All $[x,y]$ classes.
for $x \leftarrow 1$ to $\frac{k}{3}$ do

  for $y \leftarrow x$ to $k-2x-1$ do
    add $[x,y]$
  end for
end for
if $k \equiv 0(\textbf{mod}\ 3)$ then
  add $[\frac{k}{3},\frac{k}{3}]$
end if
```

At this stage, we make a few remarks about the size of equivalence classes defined by above choices of x and y.

1. $k \not\equiv 0 \pmod 3$:
   If k is not a multiple of 3, then each class contains exactly k distinct choices from the collection of $\binom{k}{3}$ choices. Hence there are $l = \frac{(k-1)(k-2)}{6}$ distinct classes of size k.

2. $k \equiv 0 \pmod 3$:

   If k is a multiple of 3, then a class of the form $[a,a,a]$ where $a = \frac{k}{3}$ contains only $\frac{k}{3}$ distinct choices. Here we get exactly one class of size $\frac{k}{3}$ and the remaining classes are of size k.

3.2.2 Configuration coverage measure $\mu_3(\mathscr{A})$

Given a length k vector v, we define the sets $D[x,y]$ correspond to each equivalence class $[x,y]$ generated by Algorithm 3.2.1 as follows:

$$D[x,y] = \bigcup_{\alpha \in G} \left\{ (v_i^\alpha, v_{i+x}^\alpha, v_{i+x+y}^\alpha) : 0 \le i \le k-1 \right\}$$

where the subscripts are taken modulo k and v_i^α stands for the image of v_i under α that is, $v_i^\alpha = v_i\alpha$. For computational convenience, we formulate the 3-way configuration coverage in terms of equivalence classes $[x,y]$ from Lemma 3.2.1 and $D[x,y]$ as follows:

$$\mu_3(A) = \frac{\sum\limits_{x,y} |[x,y]| \times \text{number of distinct 3-tuples of the form 6-21 covered in } D[x,y] + \binom{k}{3}(2^3 + g - 2)}{\binom{k}{3} g^3}$$

The second term in the numerator represents the coverage of triples of the form 1-5 by the matrices C_1 and C. Thus,

$$\mu_3(A) = \frac{\sum\limits_{x,y} |[x,y]| \times \text{number of distinct 3-tuples of the form 6-21 covered in } D[x,y]}{\binom{k}{3} g^3} + \frac{g+6}{g^3}.$$

3.3 Construction of testing arrays with high $\mu_4(\mathscr{A})$

In this section, we build several testing arrays with high 4-way configuration coverage measure for $g \ge 3$. Given fixed values of k, n, and g, so that $g-1$ is a prime or prime power, we are to construct a testing array $\mathscr{A}$ with high 4-way configuration coverage measure. Let $X = GF(g-1) \cup \{\infty\}$ be the set of g symbols on which we are to construct a testing array.

Our construction requires selecting a group G and finding a vector $v \in X^k$, called a vector with *good configuration coverage*. We use the vector $v = (v_0, v_1, \ldots, v_{k-1})$ to form a $k \times k$ circulant matrix

$$M = \begin{pmatrix} v_0 & v_{k-1} & \cdots & v_1 \\ v_1 & v_0 & \cdots & v_2 \\ \vdots & \vdots & & \vdots \\ v_{k-1} & v_{k-2} & \cdots & v_0 \end{pmatrix}.$$

Let $G = PGL(2, g-1)$. The action of G on 4-tuples from X has $g+11$ orbits. Refer Section 2.3 of Chapter 2 for the list of orbits. Vector $v = (v_0, v_1, \ldots, v_{k-1})$ is said to be a vector with good 4-way configuration coverage, if every $4 \times k$ subarray of M has a representative from most of the orbits 2-15. Let $C = \left[xJ \ : \ x \in X \right]$ be the $k \times g$ array whose columns are all-x vectors xJ where

$$J = (1, 1, \ldots, 1)^T.$$

The group $PGL(2, g-1)$ acting on M produces $g(g-1)(g-2)$ matrices that are concatenated with C to form a testing array of size $kg(g-1)(g-2)+g$ having high 4-way configuration coverage. If $kg(g-1)(g-2)+g \le n$, then

$$\left[M^{PGL(2,g-1)}, C \right]$$

is a testing array of size at most n having high 4-way configuration coverage.

3.3.1 Choice of vector v

Given a length k vector v, we define the sets $D[x,y,z]$ for positive integers x, y and z as follows:

$$D[x,y,z] = \bigcup_{\sigma \in G} \left\{ (v_i^\sigma, v_{i+x}^\sigma, v_{i+x+y}^\sigma, v_{i+x+y+z}^\sigma) : 0 \le i \le k-1 \right\}$$

where the subscripts are taken modulo k and v_i^σ stands for the image of v_i under σ. For computational convenience, we formulate the 4-way configuration coverage in terms of equivalence classes $[x,y,z]$ from Lemma 2.3.1 and $D[x,y,z]$, as follows:

$$\mu_4(\mathscr{A}) = \frac{\sum\limits_{x,y,z} |[x,y,z]| \times \text{number of distinct 4-tuples of the form 2-15 covered in } D[x,y,z] + \binom{k}{4}g}{\binom{k}{4}g^4}.$$

The second term in the numerator represents the coverage of 4-tuples of the form

$$\left\{(a,a,a,a)^T \ : \ a \in X\right\}$$

by matrix C. Here all the equivalence classes $[x,y,z]$ are obtained from Algorithm 1.

3.4 Results

We search by computer to find vectors v that produce testing arrays with high configuration coverage measure. Tables 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 3.10, 3.11, and 3.12 show vectors with high configuration coverage, the number of test cases (n) generated by our technique, and the best known size with full coverage [31]. A comparison of our constructions with the best known covering array sizes shows that our constructions produce significantly smaller testing arrays $\mathscr{A}$ with high configuration coverage. Test configuration coverage is one of the most important topics in software testing. Users would like to have some quantitative measure to judge the risk while using a product. Consider testing a software system with 40 parameters each having three values. Our construction generates a test suite with 243 test cases that ensure with probability 0.988 that software cannot fail due to interactions of 2, 3 or 4 parameters whereas the best known covering array in [31] requires 465 test cases for full coverage. The results show that the proposed methods could reduce the number of test cases significantly while compromising only slightly on the coverage.

Table 3.1: Vector v with good coverage for $t = 3$ and $g = 4$.

k	Vector v with good coverage	Our Results n (μ_3)	Best known n ($\mu_3 = 1$) [31]
35	$(0\infty_2\infty_2\infty_2 01\infty_1\infty_2\infty_1\infty_1\infty_2 11\infty_1 11\infty_1 1\infty_2 000\infty_2 011\infty_1 00$ $1\infty_1\infty_2\infty_1 01)$	96 (0.812667)	152
36	$(0\infty_2\infty_1 1\infty_1 10\infty_2 1\infty_2\infty_2\infty_2 00\infty_2\infty_1 001001\infty_2 1\infty_1\infty_1\infty_1 00$ $\infty_2 0\infty_1 11\infty_1 1)$	98 (0.815966)	152
37	$(1011101\infty_1\infty_2\infty_2\infty_2 1\infty_1 10\infty_2 00\infty_2\infty_1 0\infty_1 110\infty_1\infty_2\infty_1 11$ $\infty_1\infty_2 0\infty_1\infty_1\infty_2 1)$	100 (0.824554)	152
38	$(\infty_1 001\infty_1 1\infty_1 00\infty_1\infty_2 0\infty_1\infty_1\infty_1 111\infty_2 1\infty_2\infty_2 01001\infty_1 10$ $\infty_2\infty_1\infty_2\infty_2 011\infty_2)$	103 (0.833474)	152
39	$(\infty_1\infty_1 0\infty_2\infty_2 0\infty_1 1\infty_1\infty_2 101001\infty_2 000\infty_1\infty_1\infty_2 0\infty_2 1\infty_1 01$ $01\infty_2 10\infty_2 1\infty_1\infty_1\infty_2)$	105 (0.839705)	188
40	$(\infty_2 0\infty_2 11\infty_2\infty_2 1\infty_1 000\infty_2\infty_1 1\infty_1\infty_1\infty_2 0\infty_1\infty_2 0001\infty_2\infty_2 1$ $0\infty_1 1\infty_1 01010\infty_1\infty_1 1)$	107 (0.847166)	191
41	$(10\infty_1 0\infty_2 00\infty_1 101\infty_1 0101\infty_2 000\infty_2 001\infty_2\infty_1\infty_1\infty_2\infty_1 1\infty_2 0\infty_1 11$ $\infty_1 0\infty_1\infty_2\infty_2\infty_2)$	109 (0.85613)	191
42	$(0\infty_2\infty_1\infty_2 1\infty_1\infty_2 1\infty_2\infty_2\infty_1 101\infty_1 000001\infty_1 10\infty_2\infty_1 0\infty_1 10\infty_2 011$ $\infty_2 1\infty_1\infty_1\infty_2\infty_1 10)$	111(0.86074)	191
43	$(\infty_2\infty_1\infty_1 10\infty_2 0\infty_1\infty_2\infty_2\infty_2 011\infty_1\infty_1 10\infty_1\infty_2 00\infty_1 0\infty_1\infty_1 1000\infty_2$ $1\infty_2\infty_1 10\infty_2 1\infty_2 1011)$	113 (0.86748)	191
44	$(\infty_1\infty_2 101\infty_1\infty_2\infty_2 0\infty_2 1\infty_1\infty_1\infty_1 0\infty_1\infty_1 0\infty_2 11\infty_2\infty_1 01111\infty_2 001$ $\infty_2\infty_2 1\infty_2\infty_1 1000\infty_1 00)$	115 (0.87209)	191
45	$(000\infty_1\infty_1 01\infty_2\infty_2 0\infty_1\infty_2 000\infty_2 1\infty_2 1\infty_1\infty_2 11\infty_1 0\infty_2\infty_2\infty_1 1\infty_1\infty_1$ $\infty_2\infty_1 0010\infty_1 0\infty_2 0\infty_2 011)$	118(0.87741)	191

Table 3.2: Vector v with good coverage for $t = 3$ and $g = 4$ (continued).

k	Vector v with good coverage	Our Results n (μ_3)	Best known n ($\mu_3 = 1$) [31]
46	$(1\infty_1 10\infty_2\infty_2\infty_1 01\infty_1 1\infty_2\infty_2 1\infty_2\infty_1 11\infty_2\infty_1\infty_1 0\infty_1 0\infty_1 1100\infty_2\infty_2$ $1000\infty_1\infty_2 1\infty_2\infty_1 1\infty_2 0110)$	120 (0.88304)	192
47	$(0110\infty_2\infty_2\infty_2 01\infty_1\infty_1 00\infty_1 1\infty_1 0\infty_1 1\infty_1 1\infty_2\infty_1 10\infty_2 1\infty_2 10000\infty_1$ $\infty_2\infty_2 01\infty_2 10\infty_2\infty_1\infty_1 0\infty_1\infty_2)$	123 (0.88813)	193
48	$(\infty_1\infty_1 111\infty_2 0\infty_2 1\infty_2\infty_2 10\infty_1 0\infty_2 100\infty_2 11\infty_2\infty - 1\infty_1\infty_2 10\infty_1 10$ $\infty_1 00001\infty_1\infty_2\infty_1 10\infty_1 0\infty_1\infty_2\infty_2 0)$	125 (0.89254)	193
49	$(\infty_1 10111\infty_2\infty_1\infty_2\infty_1 00\infty_1\infty_2\infty_1\infty_1\infty_1\infty_1\infty_2 11010\infty_2\infty_1 0\infty_2\infty_2 10$ $\infty_2 100\infty_2 10\infty_1 1\infty_1\infty_2 1\infty_2\infty_1 0011)$	127 (0.89744)	194
50	$(\infty_1 11\infty_2\infty_2\infty_2 0\infty_1 011011\infty_1\infty_1\infty_2 1\infty_2 1\infty_2\infty_2 0\infty_1\infty_2 1000\infty_2 001$ $\infty_1 10\infty_2\infty_1 1101\infty_1\infty_2\infty_1\infty_1 0\infty_1\infty_1 1)$	130 (0.90202)	194
51	$(\infty_2\infty_1 0011\infty_2\infty_1 1100\infty_1\infty_2 0000\infty_1 010\infty_2 0\infty_2\infty_1\infty_2\infty_2 1\infty_1\infty_1 1\infty_1$ $\infty_1\infty_1 0\infty_2 1\infty_2 10\infty_2\infty_1 1\infty_2 010\infty_1 10)$	132 (0.90778)	212
52	$(\infty_1\infty_1\infty_2\infty_2 1\infty_2 0110\infty_1 11011\infty_2\infty_1 0\infty_1 0100\infty_2\infty_1 0\infty_2 0\infty_2 111\infty_1$ $00\infty_1\infty_1\infty_2 101\infty_2 1\infty_2\infty_2\infty_1\infty_1\infty_1 0\infty_2 1)$	134 (0.91014)	212
53	$(0\infty_2 10\infty_1 11000\infty_2\infty_2 01\infty_2 000\infty_2\infty_1 10\infty_2 0\infty_2\infty_2\infty_1 11\infty_1\infty_2\infty_1 00$ $0\infty_1\infty_1 0\infty_2 11010\infty_1\infty_2\infty_1\infty_2\infty_1\infty_1\infty_1 01)$	137 (0.91395)	212
54	$(\infty_1 0110\infty_2\infty_1\infty_1\infty_2\infty_2\infty_1\infty_1 1\infty_2 11\infty_2 10\infty_1 0\infty_1\infty_2 00\infty_1\infty_1\infty_2\infty_1\infty_1$ $0000\infty_1 1\infty_2 1\infty_2\infty_2 010101\infty_2 0\infty_1 011\infty_2 1)$	139 (0.91788)	212
55	$(\infty_2\infty_2 1\infty_1 11\infty_1 101\infty_2\infty_1\infty_2\infty_1 0\infty_2 01\infty_2 11\infty_2\infty_2 00\infty_2\infty_1 01\infty_2\infty_1$ $\infty_2\infty_1 10111\infty_1 1001\infty_1 0001\infty_2 0\infty_1 1\infty_1\infty_1\infty_1)$	141 (0.92079)	240
56	$(1000\infty_2\infty_1\infty_1\infty_2\infty_2\infty_2\infty_1\infty_2\infty_1 00\infty_2 00\infty_2 11\infty_1\infty_2\infty_2\infty_1 0110\infty_1\infty_2$ $\infty_1\infty_2 10\infty_1 0\infty_1 10\infty_1\infty_1 011\infty_2 1110\infty_2 11\infty_1 10)$	144 (0.92430)	240

Table 3.3: Vector v with good coverage for $t = 3$ and $g = 4$ (continued).

k	Vector v with good coverage	Our Results n (μ_3)	Best known n ($\mu_3 = 1$) [31]
57	$(\infty_2 0\infty_1\infty_1 1\infty_2 10\infty_2\infty_1\infty_1 0\infty_1 10\infty_1 0\infty_2\infty_2 01101\infty_2 000\infty_1\infty_1 00\infty_2$ $\infty_1 1\infty_2 00\infty_1 0\infty_2 1\infty_2 1\infty_2 1\infty_2 11001\infty_1\infty_2\infty_1\infty_1\infty_2)$	146 (0.92901)	240
58	$(\infty_1\infty_2\infty_1\infty_2 01101\infty_2 0\infty_2\infty_1\infty_2\infty_1 0\infty_2 0110010101\infty_2\infty_1\infty_2 110\infty_1 0$ $\infty_2\infty_2 0\infty_1\infty_2 00\infty_2\infty_1 11111\infty_1\infty_2\infty_1\infty_1 11\infty_1\infty_1 1)$	149 (0.93203)	240
59	$(0\infty_2 1110\infty_2\infty_2\infty_1 0\infty_1 10000\infty_2 0\infty_1\infty_1 1\infty_1 0\infty_2 0\infty_1 11\infty_1\infty_2\infty_2\infty_1 1$ $\infty_2\infty_2 0101\infty_1 1\infty_2\infty_2\infty_1\infty_2 0\infty_2\infty_1\infty_1\infty_2 011\infty_1 01001)$	151 (0.93387)	240
60	$(00\infty_2\infty_1 010\infty_2\infty_2\infty_2\infty_1 11\infty_1\infty_1\infty_1 00\infty_1\infty_2\infty_2 1\infty_1 0\infty_1 01\infty_2 00\infty_2 1$ $1\infty_2 01\infty_1\infty_2\infty_1\infty_2 110\infty_2 10\infty_2 1101\infty_1\infty_1\infty_1\infty_2 1\infty_1 110)$	153 (0.93512)	240

Table 3.4: Vector v with good coverage for $t = 3$ and $g = 5$.

k	Vector v with good coverage	Our Results n (μ_3)	Best known n ($\mu_3 = 1$) [31]
15	$(\infty_2\infty_2 1\infty_2 0021\infty_1 1\infty_2 2\infty_1 12)$	110 (0.68035)	245
16	$(011\infty_2\infty_2 021\infty_1 12\infty_2\infty_1 2\infty_2 0)$	116 (0.70102)	245
17	$(0\infty_2\infty_2 1\infty_1 021\infty_1 010\infty_2\infty_1 220)$	123 (0.72640)	245
18	$(0221\infty_2\infty_1 101\infty_2 2\infty_1\infty_1 0\infty_1\infty_1 20)$	129 (0.73917)	245
19	$(\infty_1\infty_1 1\infty_1 1001211\infty_2 2\infty_1\infty_1\infty_2 220)$	135 (0.76094)	245
20	$(\infty_2\infty_2 2\infty_1 2012\infty_1 01111\infty_1 2\infty_1\infty_2 20)$	141 (0.77431)	245

Table 3.5: Vector v with good coverage for $t = 3$ and $g = 5$ (continued).

k	Vector v with good coverage	Our Results n (μ_3)	Best known n ($\mu_3 = 1$) [31]
21	$(\infty_2 1210\infty_1 2\infty_1 00002\infty_1\infty_1 2\infty_2\infty_1 010)$	148 (0.79082)	245
22	$(\infty_2\infty_1 0\infty_2 221\infty_2 0011\infty_1 21\infty_2\infty_2\infty_2 2020)$	154 (0.80731)	245
23	$(1\infty_2 211\infty_2 12120\infty_2\infty_2 0022\infty_1 1\infty_2\infty_1 0\infty_1)$	161 (0.82233)	245
24	$(\infty_2 222\infty_2\infty_2\infty_1\infty_2 1\infty_1 012\infty_2 01\infty_1 01\infty_2 1200)$	168 (0.83266)	245
25	$(\infty_2\infty_1 2\infty_2\infty_2 201\infty_1\infty_2 0\infty_2 0\infty_1 21210\infty_2 11122)$	174 (0.84556)	249
26	$(1\infty_1\infty_1 1\infty_2 2\infty_1\infty_1\infty_2 212220\infty_1 11\infty_1\infty_2 01002\infty_1)$	181 (0.85408)	249
27	$(\infty_2 21\infty_1\infty_1\infty_1 2010\infty_1 0220\infty_1 111\infty_1 0\infty_2 1\infty_1\infty_2 21)$	188 (0.86619)	249
28	$(\infty_2 2\infty_2\infty_1 0\infty_2 10\infty_1\infty_1 121000\infty_2\infty_2 12\infty_2 12221\infty_2 0)$	194 (0.87241)	249
29	$(\infty_2 21\infty_1 2022\infty_2 1\infty_1 011\infty_1\infty_1\infty_1 000211\infty_2 2\infty_1\infty_2 12)$	200 (0.88190)	249
30	$(02\infty_1 0\infty_1 01\infty_2\infty_1 1\infty_2 2210\infty_2\infty_1\infty_1 1\infty_2\infty_2 20222\infty_2 200)$	206 (0.88969)	249
31	$(\infty_2\infty_2\infty_1 0\infty_1 0\infty_1 2121\infty_1\infty_2 1\infty_2\infty_2 2001\infty_2 110211\infty_1 010)$	213 (0.89605)	249
32	$(\infty_2\infty_1 220\infty_1 210\infty_1 1\infty_1 2\infty_1 110111\infty_2 0002\infty_2 1\infty_1\infty_1 2\infty_1\infty_2)$	219 (0.90523)	349
33	$(1\infty_1 01\infty_2\infty_1 0\infty_2 02\infty_2 2\infty_2 100\infty_2\infty_2\infty_1\infty_1\infty_1 202\infty_2 2211\infty_1 012)$	225 (0.91406)	365
34	$(\infty_2\infty_2 2\infty_2\infty_2 2\infty_1 1\infty_1 1\infty_2\infty_1 01211\infty_1\infty_2 1\infty_2 22002021\infty_1\infty_2 021)$	231 (0.91981)	365

Table 3.6: Vector v with good coverage for $t = 3$ and $g = 5$ (continued).

k	Vector v with good coverage	Our Results n (μ_3)	Best known n ($\mu_3 = 1$) [31]
35	$(01121\infty_1 22\infty_2 1000\infty_1\infty_2 002\infty_1 0\infty_1 01\infty_1 12\infty_2\infty_2 0\infty_1 1$ $\infty_2\infty_1\infty_1 0)$	237 (0.92222)	365
36	$(\infty_2 2111\infty_2 0122\infty_1 2\infty_2\infty_2 02\infty_2 20021\infty_1 0\infty_1\infty_2\infty_2 11\infty_1\infty_2$ $\infty_1 0\infty_2 00)$	243 (0.92755)	365
37	$(122101\infty_2\infty_2 10\infty_2 1\infty_2\infty_2\infty_1 2\infty_2 222\infty_2\infty_1\infty_1 12\infty_2 0\infty_2 21$ $\infty_1 2\infty_1 1000)$	249 (0.93782)	365
38	$(1\infty_2\infty_1 0220011\infty_1 2\infty_1 2121\infty_2\infty_2\infty_1 202\infty_1\infty_2 22\infty_1 0\infty_2$ $20\infty_1 011\infty_2\infty_1)$	255 (0.94118)	365
39	$(2\infty_1\infty_2\infty_1 2120\infty_2 102\infty_2\infty_2\infty_2 020\infty_2 0220111\infty_2 12\infty_2 0$ $\infty_1 1\infty_1\infty_2 00\infty_2 1)$	262 (0.94264)	365
40	$(20\infty_1 1\infty_2\infty_2 0\infty_1\infty_2 20\infty_2 2\infty_2\infty_2\infty_1 21\infty_1\infty_1 1121000$ $\infty_1 1\infty_1 2010\infty_1 0\infty_1 102)$	268 (0.94694)	365

Table 3.7: Vector v with good coverage for $t = 4$ and $g = 3$.

k	Vector v with good coverage	Our Results n (μ_4)	Best known n ($\mu_4 = 1$) [31]
16	$(00001001\infty\infty 011\infty 1\infty)$	99 (0.828)	237
17	$(0000010\infty\infty 101\infty 01\infty 1)$	105 (0.851)	282
18	$(00010\infty 0\infty 1001\infty 111\infty\infty)$	111 (0.864)	293
19	$(000010010\infty 01\infty 0\infty 111\infty)$	117 (0.883)	305

Table 3.8: Vector v with good coverage for $t = 4$ and $g = 3$ (continued).

k	Vector v with good coverage	Our Results n (μ_4)	Best known n ($\mu_4 = 1$) [31]
20	$(0000110101\infty0\infty10\infty\infty11\infty)$	123 (0.892)	314
21	$(00001010\infty1\infty\infty10\infty\infty001\infty1)$	129 (0.906)	315
22	$(0000011\infty0\infty0110\infty1\infty\infty\infty01\infty)$	135 (0.913)	315
23	$(0000001\infty\infty0101\infty10\infty10\infty\infty\infty1)$	141 (0.923)	315
24	$(00000001\infty\infty0101\infty10\infty101\infty\infty1)$	147 (0.924)	315
25	$(0000000011\infty0\infty011\infty01\infty0\infty11\infty)$	153 (0.930)	363
28	$(1\infty\infty00\infty\infty1\infty01101111\infty0\infty0101\infty\infty\infty1)$	171 (0.957)	383
29	$(010\infty00\infty1\infty0\infty\infty\infty101\infty00\infty000111\infty10)$	177 (0.961)	392
30	$(011\infty11\infty\infty\infty001\infty\infty\infty1\infty10\infty\infty0\infty1100\infty01)$	163 (0.969)	393
35	$(01\infty0\infty\infty1000\infty01\infty\infty0\infty1\infty111\infty\infty\infty01\infty01000\infty1)$	213 (0.979)	429
36	$(11\infty0110\infty\infty00\infty111101011\infty001\infty\infty\infty\infty\infty100\infty0\infty)$	219 (0.981)	441
38	$(1\infty1\infty111\infty\infty010\infty10\infty\infty00010\infty\infty0\infty\infty\infty1101\infty\infty100\infty)$	231 (0.985)	447
39	$(001\infty\infty11\infty11\infty0001\infty11\infty101\infty\infty\infty1\infty0\infty0010\infty00\infty\infty0)$	237 (0.986)	453
40	$(100\infty\infty00001\infty\infty1\infty10\infty000\infty\infty\infty0\infty10\infty\infty1\infty1\infty0111\infty01)$	243 (0.988)	465

Table 3.9: Vector v with good coverage for $t = 4$ and $g = 4$.

k	Vector v with good coverage	Our Results n (μ_4)	Best known n ($\mu_4 = 1$) [31]
18	$(00010021\infty\infty\infty21020\infty2)$	436 (0.851)	760
19	$(0000121011\infty01\infty0\infty221)$	460 (0.866)	760
20	$(0000112101202\infty0221\infty2)$	484 (0.878)	760
21	$(0000011021010\infty2\infty0221\infty)$	508 (0.887)	1012
22	$(0000001102\infty02021\infty\infty01\infty1)$	532 (0.894)	1012
23	$(00000001210210\infty\infty20112\infty1)$	556 (0.898)	1012
24	$(00000000121\infty011\infty02\infty0\infty112)$	580 (0.899)	1012
25	$(000000000121220\infty011\infty2012\infty)$	604 (0.901)	1012
26	$(00100\infty2221110102\infty0022\infty020\infty2)$	628 (0.921)	1012
27	$(0100\infty2221110102\infty0022\infty020\infty2)$	652 (0.928)	1012
28	$(01110\infty0102\infty021110022001\infty1001)$	676 (0.933)	1012
29	$(0\infty\infty122101\infty000220200221220\infty02)$	702 (0.937)	1012
30	$(10\infty20\infty020\infty2\infty2\infty01\infty2222\infty022002\infty1)$	726 (0.943)	1012

Table 3.10: Vector v with good coverage for $t = 4$ and $g = 5$.

k	Vector v with good coverage	Our Results n (μ_4)	Best known n ($\mu_4 = 1$) [31]
21	$(110131300\infty30010\infty\infty3203)$	1265 (0.834)	1865
22	$(3\infty32011200\infty\infty00\infty0\infty10010)$	1325 (0.842)	1865
23	$(0002\infty03100\infty203021332320)$	1385 (0.854)	1865
24	$(003\infty21022212300032302310)$	1445 (0.860)	1865
25	$(\infty200\infty0\infty\infty31020\infty300303\infty\infty33)$	1505 (0.869)	2485
26	$(202002211000\infty0121031\infty\infty2300)$	1565 (0.873)	2485
27	$(\infty\infty03002030\infty000\infty11\infty0031301\infty3)$	1625 (0.880)	2485
28	$(013333130320\infty1\infty1003200310300)$	1685 (0.883)	2485
29	$(00012212\infty010\infty3110031020031010)$	1745 (0.891)	2485
30	$(33001\infty0\infty000330\infty\infty010012\infty1313001)$	1805 (0.894)	2485
31	$(033\infty21333010313\infty303320030012020)$	1865 (0.895)	2485
32	$(310031000\infty330130321\infty\infty03031111310)$	1925 (0.897)	2485
33	$(\infty0010\infty\infty3\infty0\infty2\infty01\infty00\infty12222\infty\infty03\infty020\infty)$	1985 (0.904)	2485
34	$(\infty\infty3\infty00101001\infty0\infty001\infty002\infty01110231112)$	2045 (0.906)	2485

Table 3.11: Vector v with good coverage for $t = 4$ and $g = 5$ (continued).

k	Vector v with good coverage	Our Results n (μ_4)	Best known n ($\mu_4 = 1$) [31]
35	$(1203003303\infty0\infty013233310\infty032020003220)$	2105 (0.906)	2485
36	$(12022\infty32032300232223220001010200\infty2230)$	2165 (0.912)	2485

Table 3.12: Vector v with good coverage for $t = 4$ and $g = 6$.

k	Vector v with good coverage	Our Results n (μ_4)	Best known n ($\mu_4 = 1$) [31]
25	$(00040301400303340432 0\infty1\infty\infty)$	3006 (0.811)	6325
26	$(\infty0\infty40021404010013010011444)$	3126 (0.819)	6456
27	$(433\infty\infty01\infty\infty20\infty03020\infty\infty0\infty00401\infty)$	3246 (0.826)	6606
28	$(4023031100232200\infty21\infty\infty2020020)$	3366 (0.829)	6714
29	$(00\infty40023103301343401230334400)$	3486 (0.834)	6852
30	$(1\infty\infty\infty42\infty4040004\infty104\infty03034\infty\infty0300)$	3606 (0.836)	6966
31	$(44122002\infty2000020202031\infty42044001)$	3726 (0.838)	7092
32	$(44441341\infty424000\infty\infty040004410103400)$	3846 (0.846)	7200
33	$(0330344\infty02321331003130000 30\infty4303\infty)$	3966 (0.855)	7320

Chapter 4

Covering Arrays on Product Graphs

Covering arrays are used to design test suites for software and hardware testing. In such an application, each row of the array corresponds to a parameter in the system, each column corresponds to a test case and the symbols correspond to the values of the parameters. Such a test suite covers each possible parameter-value configuration for any pair of parameters. In software testing, we may know in advance that two specific parameters do not interact. Then it is not necessary that each possible parameter-value configuration for these two parameters be covered. We can use a graph structure to describe which pairs of parameters need to be covered.

A covering array on a graph G with alphabet size g, denoted by $CA(n,G,g)$, is a $|V(G)| \times n$ array on $\mathbb{Z}_g$ with the property that any two rows which correspond to adjacent vertices in G are qualitatively independent. Given a graph G and a positive integer g, a covering array on G with minimum size is called optimal. There have been some studies of covering arrays on graphs. Seroussi and Bshouty [90] proved that the problem of finding the smallest binary covering array on a graph is NP-hard problem. Covering arrays on graphs have been introduced in the conclusion of Stevens's Ph.D. thesis [93]. Meagher and Stevens studied covering arrays on graphs in detail in [68].

Our primary concern in this chapter is with constructions that make optimal covering arrays on large graphs that are obtained from product of smaller graphs. See also [3]. In Section 4.1, we recall some basic definitions and results from graph theory which will be used in this chapter and Chapter 5. We review in Section 4.2 some known results on covering arrays on graphs. In Section 4.3, we consider four most extensively studied graph products in the literature. In Section 4.4 and Section 4.5, we give upper and lower bounds on the size of an optimal covering array on a product graph. In Section 4.6, we find families of graphs for which the size of a covering array on a product graph achieves the lower bound with respect to the Cartesian product. Finally, in Section 4.7, we present a polynomial time approximation algorithm with approximation ratio $\log(\frac{|V|}{2^{k-1}})$ for constructing covering arrays on graphs having k prime factors with respect to the Cartesian product.

4.1 Graph Theory

In this section, we recall some definitions and relevant results in graph theory from [103].

A *graph* G is a pair $G = (V, E)$ where V is a set of *vertices* and E is a set of unordered pairs of vertices. The elements of E are called *edges*. We write $V(G)$ for the set of vertices and $E(G)$ for the set of edges of a graph G. A *loop* is an edge vv for some $v \in V(G)$. A graph G is *simple* if it has no loops and $E(G)$ is not a multi-set. A graph is *finite* if its vertex set is finite. Throughout this chapter, we will only consider finite simple graphs. Two vertices u and v are *adjacent* in G, if $uv \in E(G)$.

4.1.1 Walks, Paths and Cycles

A *walk* in a simple graph G is a sequence of vertices $v_0, v_1, \dots, v_k$, where $v_i v_{i+1} \in E(G)$. A walk is a *path* if all v_i are distinct. If for such a path with $k \geq 2$, $v_0 v_k$ is also an edge in G, then $v_0, v_1, \dots, v_k, v_0$ is a *cycle*. The *length* of a path, cycle or walk is the number of edges in it. We denote the path of length k by P_k and the cycle of length k by C_k. A graph having no cycle is *acyclic*.

A graph is *connected* when there is a path between every pair of vertices. A graph that is not connected is *disconnected*. In a disconnected graph G, a *connected component* is a maximal connected subgraph of G. A *spanning subgraph* is a subgraph that contains all the vertices of the original graph.

4.1.2 Trees

A *tree* is a connected acyclic graph. A *rooted tree* is a tree with a designated vertex called the *root*. In a rooted tree, the *level* of a vertex v is the length of the unique path from the root to v. Thus root has level 0. If vertex v immediately precedes vertex w on the path from the root to w, then v is the *parent* of w, and w is a *child* of v. The *height* of a rooted tree is the length of the longest path from the root. In a rooted tree, a *leaf* is any vertex having no children and an *internal vertex* is any vertex that has at least one child.

Definition 4.1.1. A *binary tree* is a rooted tree in which every vertex has at most two children.

Example 4.1.1. Figure 4.1 shows a binary tree of height 3. The root of this binary tree is v_0.

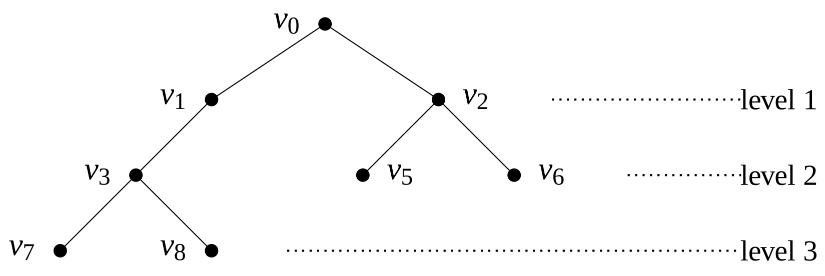


Figure 4.1: A binary tree T.

4.1.3 Graph homomorphism and isomorphism

We now recall the definitions of graph homomorphism and isomorphism used here; for more information see [50].

Definition 4.1.2. A *homomorphism* from G to H is a map $\varphi : V(G) \to V(H)$ that preserves adjacency: if uv is an edge in G, then $\varphi(u)\varphi(v)$ is an edge in H.

We say $G \to H$ if there is a homomorphism from G to H, and $G \equiv H$ if $G \to H$ and $H \to G$. A *weak homomorphism* from G to H is a map $\varphi : V(G) \to V(H)$ such that if uv is an edge in G, then either $\varphi(u)\varphi(v)$ is an edge in H, or $\varphi(u) = \varphi(v)$. Clearly every homomorphism is automatically a weak homomorphism.

Definition 4.1.3. Two graphs G and H are said to be *isomorphic* if there is a bijection mapping φ from the vertex set $V(G)$ to the vertex set $V(H)$ such that $uv \in E(G)$ if and only if $\varphi(u)\varphi(v) \in E(H)$. The mapping φ is called an isomorphism. An isomorphism preserves adjacency as well as non-adjacency of vertices.

A homomorphism from a graph G to itself is an *endomorphism*. An isomorphism from a graph G to itself is an *automorphism*. The set of all automorphisms of a graph G forms a group, denoted $Aut(G)$, the *automorphism group of G*.

4.1.4 Colourings and Cliques

A *complete graph* is a graph in which each pair of vertices is adjacent. A complete graph on n vertices is denoted by K_n.

Definition 4.1.4. A *proper colouring* on a graph is an assignment of colours to each vertex such that adjacent vertices receive a different colour. The *chromatic number* of a graph G, $\chi(G)$, is defined to be the size of the smallest set of colours such that a proper colouring exists with that set.

In terms of graph homomorphism, a proper colouring of a graph G with n colours is equivalent to a homomorphism from G to K_n. The chromatic number $\chi(G)$ of a graph G is the smallest n such that $G \to K_n$.

Definition 4.1.5. A *maximum clique* in a graph G is a maximum set of pairwise adjacent vertices. The *maximum clique number* of a graph G, denoted $\omega(G)$, is defined to be the size of a maximum clique.

If G has a clique of size n, then there is a homomorphism $K_n \to G$ and the maximum clique number of G is the largest n for which $K_n \to G$. For graphs G and H, if there is a homomorphism $G \to H$ then $\chi(G) \leq \chi(H)$ and $\omega(G) \leq \omega(H)$. Also, the chromatic number of a graph is always greater than or equal to its clique number.

4.1.5 Bipartite graphs and Matchings

Definition 4.1.6. A *bipartite graph* is a graph whose vertices can be divided into two disjoint sets V_1 and V_2 such that every edge connects a vertex in V_1 to one in V_2.

Every bipartite graph admits a homomorphism to K_2. Hence bipartite graphs are 2-colourable. Trees are examples of bipartite graphs. A *complete bipartite graph* is a special kind of bipartite graph where every vertex of V_1 is adjacent to every vertex of V_2. A complete bipartite graph with $|V_1| = m$ and $|V_2| = n$ is denoted $K_{m,n}$.

Definition 4.1.7. A *matching M* in a graph G is a family of pairwise disjoint edges. A *perfect matching* or *1-factor* is a matching that saturates every vertex.

We now state some known results about matchings in bipartite graphs, which we rely on in the coming sections. The maximum degree of a graph G is denoted $\Delta(G)$. If M and M' are two matchings of G, then the *symmetric difference*

$$M\Delta M' = (M - M') \cup (M' - M).$$

Proposition 4.1.1. Every component of the symmetric difference of two matchings is a path or an even cycle.

Proposition 4.1.2. A bipartite graph G with maximum degree $\Delta(G)$ is union of $\Delta(G)$ matchings.

4.2 Covering arrays on graphs

Meagher and Stevens introduce and study covering arrays on graphs in [68, 66]. We recall some of their results here.

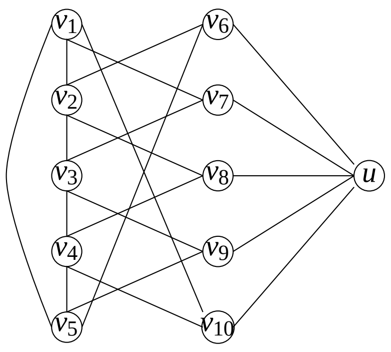
Definition 4.2.1. A *covering array on a graph* G with alphabet size g and $k = |V(G)|$ is a $k \times n$ array on $\mathbb{Z}_g$. Each row in the array corresponds to a vertex in the graph G. The array has the property that any two rows which correspond to adjacent vertices in G are qualitatively independent.

A covering array on a graph G will be denoted by $CA(n,G,g)$. The smallest possible covering array on a graph G will be denoted

$$CAN(G,g) = \min_{n\in\mathbb{N}} \{n \ : \ \text{there exists a } CA(n,G,g)\}$$

Given a graph G and a positive integer g, a covering array on G with minimum size is called *optimal*.

Example 4.2.1. An optimal binary covering array on a graph G with $\chi(G) = 4$ and $\omega(G) = 2$ is shown in Figure 4.2.



$$\begin{array}{c}
\mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \\ \mathbf{v_6} \\ \mathbf{v_7} \\ \mathbf{v_8} \\ \mathbf{v_9} \\ \mathbf{v_{10}} \\ \mathbf{u}
\end{array}
\begin{bmatrix}
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 1 & 1 & 0
\end{bmatrix}$$

Figure 4.2: A graph G and an optimal covering array $CA(5,G,2)$.

Let g and n be two positive integers where $n \geq g^2$. A *qualitatively independent graph* $QI(n,g)$ is a graph where the vertices are all length n vectors over $\mathbb{Z}_g$, in which each alphabet occurs at least g times, and the vectors have 0 in their first position. Two vertices are adjacent if their corresponding vectors are qualitatively independent.

Meagher and Stevens [68] proved that given a graph G and non-negative integers g and n, there exists a $CA(n,G,g)$ if and only if there exists a graph homomorphism $G \to QI(n,g)$.

Consider the graph G and the covering array on G, $C = CA(5,G,2)$ in Figure 4.2. We now prove that C is an optimal covering array on G. For the sake of contradiction, suppose there exists a covering array on G of size 4. There exists a $CA(4,G,2)$ if and only if there exists a graph homomorphism

$$G \to QI(4,2).$$

$QI(4,2)$ is shown in Figure 4.3. As $QI(4,2)$ is isomorphic to K_3, we get

$$G \to QI(4,2) \to K_3.$$

This implies $\chi(G) \leq 3$, which is a contradiction to the fact that $\chi(G) = 4$. Thus $CAN(G,2) = 5$.

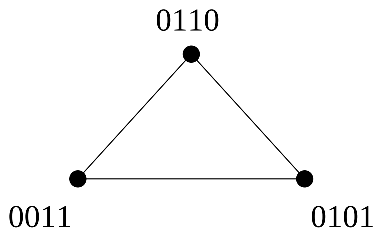


Figure 4.3: The qualitatively independent graph $QI(4,2)$.

A detailed study of $QI(n,g)$ especially for $g = 2$ is given in [66]. The lemma, given below will be used in Theorem 4.4.2.

Lemma 4.2.1. [68] *Let G and H be graphs. If $G \to H$ then $CAN(G,g) \leq CAN(H,g)$.*

4.3 Graph products

In this section, we give several definitions of graph products from [46] that we use in this chapter. A graph product is a binary operation on the set of all finite graphs. However, among all possible associative graph products, the most extensively studied in the literature are the Cartesian product, the direct product, the strong product and the lexicographic product.

Definition 4.3.1. The *Cartesian product* of graphs G and H, denoted by $G\Box H$, is the graph with

$$V(G\Box H) = \{(g,h)|g \in V(G) \text{ and } h \in V(H)\},$$
$$E(G\Box H) = \{(g,h)(g',h')|g = g', hh' \in E(H), \text{ or } gg' \in E(G), h = h'\}.$$

The graphs G and H are called the *factors* of the product $G\Box H$.

In general, given graphs $G_1, G_2, ..., G_k$, then $G_1\Box G_2\Box\cdots\Box G_k$, is the graph with vertex set $V(G_1)\times V(G_2)\times\cdots\times V(G_k)$, and two vertices $(u_1,u_2,\ldots,u_k)$ and $(v_1,v_2,\ldots,v_k)$ are adjacent if and only if $u_iv_i \in E(G_i)$ for exactly one index $1 \le i \le k$ and $u_j = v_j$ for each index $j \ne i$.

Example 4.3.1. Let $G = P_2$ and $H = P_3$. Then their Cartesian product $P_2\Box P_3$ is the graph shown in Figure 4.4.

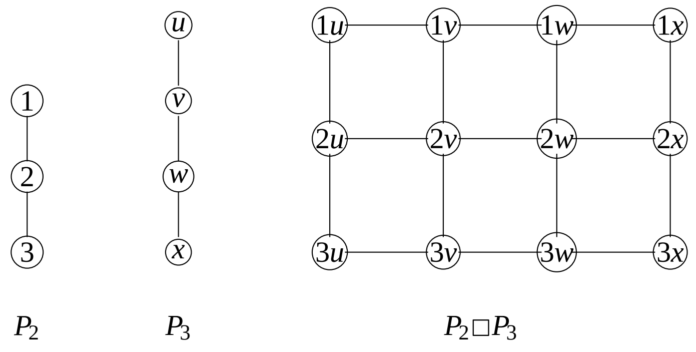


Figure 4.4: The Cartesian product of P_2 and P_3.

Definition 4.3.2. The *direct product* of graphs $G_1, G_2, ..., G_k$, denoted by $G_1\times G_2\times\cdots\times G_k$, is the graph with vertex set $V(G_1)\times V(G_2)\times\cdots\times V(G_k)$, and for which vertices $(u_1,u_2,...,u_k)$ and $(v_1,v_2,...,v_k)$ are adjacent precisely if $u_iv_i \in E(G_i)$ for each index i.

Example 4.3.2. The direct product of graphs P_2 and P_3 is shown in Figure 4.5.

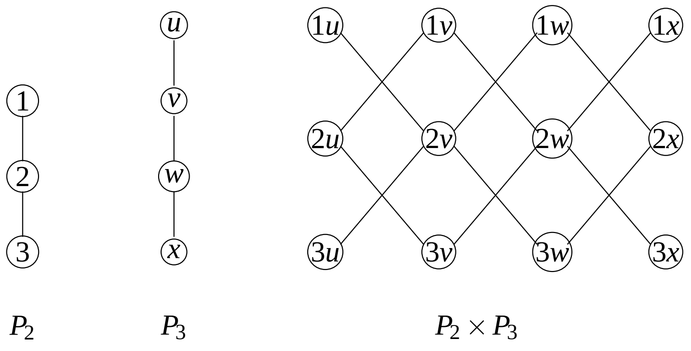


Figure 4.5: The direct product of P_2 and P_3.

Definition 4.3.3. The *strong product* of graphs $G_1, G_2, ..., G_k$, denoted by $G_1 \boxtimes G_2 \boxtimes \cdots \boxtimes G_k$, is the graph with vertex set $V(G_1) \times V(G_2) \times \cdots \times V(G_k)$, and distinct vertices $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$ are adjacent if and only if either $u_iv_i \in E(G_i)$ or $u_i = v_i$ for each $1 \le i \le k$. We note that in general $E(\boxtimes_{i=1}^{k} G_i) \neq E(\square_{i=1}^{k} G_i) \cup E(\times_{i=1}^{k} G_i)$, unless $k = 2$.

Example 4.3.3. The strong product of graphs P_2 and P_3 is shown in Figure 4.6.

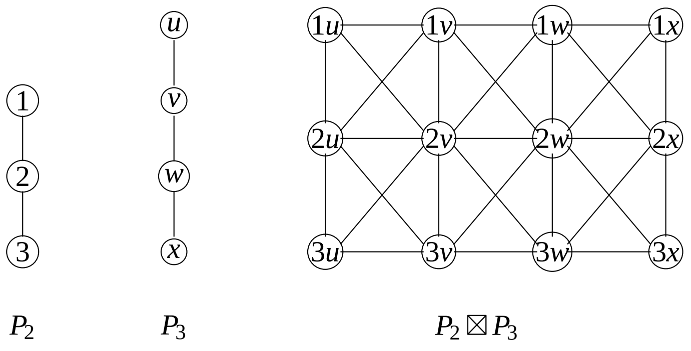


Figure 4.6: The strong product of P_2 and P_3.

Definition 4.3.4. The *lexicographic product* of graphs $G_1, G_2, ..., G_k$, denoted by $G_1 \circ G_2 \circ \cdots \circ G_k$, is the graph with vertex set $V(G_1) \times V(G_2) \times \cdots \times V(G_k)$, and two vertices $(u_1, u_2, ..., u_k)$ and $(v_1, v_2, ..., v_k)$ are adjacent if and only if for some index $j \in \{1, 2, ..., k\}$ we have $u_jv_j \in E(G_j)$ and $u_i = v_i$ for each index $1 \le i < j$.

Example 4.3.4. The lexicographic product of graphs P_2 and P_3 is shown in Figure 4.7.

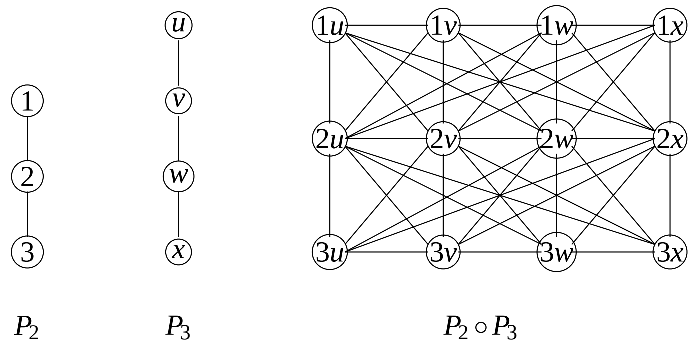


Figure 4.7: The lexicographic product of P_2 and P_3.

A graph is *prime* with respect to a given graph product if it is nontrivial and cannot be represented as the product of two non-trivial graphs. For the Cartesian product, it means that a non-trivial graph G is prime if $G = G_1 \Box G_2$ implies that either G_1 or G_2 is K_1. Similar observation is true for other three products. The uniqueness of the prime factor decomposition of connected graphs with respect to the Cartesian product was first shown by Subidussi (1960) [85], and independently by Vizing (1963) [100]. Prime factorization is not unique for the Cartesian product in the class of possibly disconnected simple graphs [46]. It is known that any connected graph factors uniquely into prime graphs with respect to the Cartesian product.

Theorem 4.3.1. [85, 100] *Every connected graph has a unique representation as a product of prime graphs with respect to the Cartesian product, up to isomorphism and the order of the factors. The number of prime factors is at most* $\log_2 |V|$.

For any connected graph $G = (V,E)$, the prime factors of G with respect to the Cartesian product can be computed in $O(|E|\log|V|)$ times and $O(|E|)$ space. For more details see [46, Chapter 23].

4.4 Upper bounds on $CAN(G_1 * G_2)$

Let $*$ represent either the Cartesian, the direct, the strong, or the lexicographic product operation. Given covering arrays $CA(n_1, G_1, g)$ and $CA(n_2, G_2, g)$, one can construct covering array on $G_1 * G_2$ as follows: the row corresponds to the vertex (u, v) is obtained by horizontally concatenating the row corresponds to the vertex u in $CA(n_1, G_1, g)$ with the row corresponds to the vertex v in $CA(n_2, G_2, g)$. Hence an obvious upper bound for the covering array number is given by

$$CAN(G_1 * G_2, g) \leq CAN(G_1, g) + CAN(G_2, g)$$

We now propose some improvements of this bound. A column of a covering array is *constant* if, for some symbol v, every entry in the column is v. In a *standardized* $CA(n, G, g)$ the first column is constant. Because symbols within each row can be permuted independently, if a $CA(n, G, g)$ exists, then a standardized $CA(n, G, g)$ exists.

Theorem 4.4.1. *Let $G = G_1 \boxtimes G_2$, and $g \geq 2$ be a positive integer. Suppose for each $1 \leq i \leq 2$ there exists a $CA(n_i, G_i, g)$, then there exists a $CA(n, G, g)$ where $n = n_1 + n_2 - 2$. Hence, $CAN(G, g) \leq CAN(G_1, g) + CAN(G_2, g) - 2$.*

Proof. Without loss of generality, we assume that for each $1 \leq i \leq 2$, the first column of $CA(n_i, G_i, g)$ is a constant column on symbol $i - 1$. Let C_i be the array obtained from $CA(n_i, G_i, g)$ by removing the first column. Form an array A with $|V(G_1)| \times |V(G_2)|$ rows and $n_1 + n_2 - 2$ columns, indexing rows as (u, v) for $1 \leq u \leq |V(G_1)|$ and $1 \leq v \leq |V(G_2)|$. Row (u, v) is obtained by horizontally concatenating the row u of C_1 with the row v of C_2. Consider two rows of A that correspond to two adjacent vertices (u_1, v_1) and (u_2, v_2) of G. Two distinct vertices (u_1, v_1) and (u_2, v_2) are adjacent in G if and only if either $u_1u_2 \in E(G_1)$ and $v_1 = v_2$ or $u_1 = u_2$ and $v_1v_2 \in E(G_2)$ or $u_1u_2 \in E(G_1)$ and $v_1v_2 \in E(G_2)$. Without loss of generality, suppose $u_1u_2 \in E(G_1)$. These two rows (u_1, v_1) and (u_2, v_2) of A cover all pairs except probably the pair $(0, 0)$ in the first $n_1 - 1$ columns when $u_1u_2 \in E(G_1)$. When $v_1 = v_2$ or $v_1v_2 \in E(G_2)$ the pair $(0, 0)$ is covered in the last $n_2 - 1$ columns. $\square$

Corollary 4.4.1. *Let $G = G_1 \boxtimes G_2 \boxtimes \cdots \boxtimes G_k$, $k \geq 2$ and g be a positive integer. Suppose for each $1 \leq i \leq k$ there exists a $CA(n_i, G_i, g)$, then there exists a $CA(n, G, g)$ where $n = \sum_{i=1}^{k} n_i - 2k + 2$. Hence, $CAN(G, g) \leq \sum_{i=1}^{k} CAN(G_i, g) - 2k + 2$.*

Proof. Since the strong product of graphs is an associative binary operation,

$$G = ((\cdots((G_1 \boxtimes G_2) \boxtimes G_3) \boxtimes \cdots) \boxtimes G_k).$$

A covering array on G of size $\sum_{i=1}^{k} CAN(G_i, g) - 2(k-1)$ is derived by iterating Theorem 4.4.1 $k-1$ times. □

Using the definition of strong product of graphs, we have the following result as a corollary.

Corollary 4.4.2. *Let $G = G_1 * G_2 * \cdots * G_k$, $k \geq 2$ and g be a positive integer, where $* \in \{\Box, \times\}$. Then, $CAN(G, g) \leq \sum_{i=1}^{k} CAN(G_i, g) - 2k + 2$.*

Theorem 4.4.2. *Let $G = G_1 \times G_2 \times \cdots \times G_k$, $k \geq 2$ and g be a positive integer. Suppose for each $1 \leq i \leq k$ there exists a $CA(n_i, G_i, g)$. Then there exists a $CA(n, G, g)$ where $n = \min_i n_i$. Hence, $CAN(G, g) \leq \min_i CAN(G_i, g)$.*

Proof. Without loss of generality assume that $n_1 = \min_i n_i$. It is known that $G_1 \times G_2 \times \cdots \times G_k \rightarrow G_1$. Using Lemma 4.2.1, we have $CAN(G, g) \leq CAN(G_1, g)$. □

All the above-mentioned bounds are based on recursive constructions of covering arrays on product graphs using covering arrays on factor graphs. We now use graph homomorphism to find bounds on $CAN(G, g)$.
Since there are homomorphisms between the following graphs

$$K_{\omega(G)} \rightarrow G \rightarrow K_{\chi(G)},$$

we can find bound on the size of a covering array on a graph from the graph's chromatic number and clique number. For all graphs G,

$$CAN(K_{\omega(G)}, g) \leq CAN(G, g) \leq CAN(K_{\chi(G)}, g).$$

We have the following results on proper colouring of product graphs [84]

$$\chi(G_1 \Box G_2) = \max\{\chi(G_1), \chi(G_2)\}.$$

For other graph products there are no explicit formulae for chromatic number but following bounds are mentioned in [46].

$$\chi(G_1 \times G_2) \leq \min\{\chi(G_1), \chi(G_2)\}$$

$$\chi(G_1 \boxtimes G_2) \leq \chi(G_1 \circ G_2) \leq \chi(G_1)\chi(G_2).$$

A proper colouring of $G_1 * G_2$ with $\chi(G_1 * G_2)$ colours is equivalent to a homomorphism from $G_1 * G_2$ to $K_{\chi(G_1 * G_2)}$ for any $* \in \{\Box, \times, \boxtimes, \circ\}$. Hence

$$CAN(G_1 \Box G_2, g) \leq CAN(K_{\max\{\chi(G_1),\chi(G_2)\}}, g)$$

$$CAN(G_1 \times G_2, g) \leq CAN(K_{\min\{\chi(G_1),\chi(G_2)\}}, g)$$

$$CAN(G_1 \boxtimes G_2, g) \leq CAN(K_{\chi(G_1)\chi(G_2)}, g)$$

$$CAN(G_1 \circ G_2, g) \leq CAN(K_{\chi(G_1)\chi(G_2)}, g).$$

4.5 Lower bounds on $CAN(G_1 * G_2)$

Note that $G_1 \to G_1 * G_2$ and $G_2 \to G_1 * G_2$ for $* \in \{\Box, \boxtimes, \circ\}$ give:

$$\max\{CAN(G_1, g), CAN(G_2, g)\} \leq CAN(G_1 * G_2, g) \tag{4.1}$$

We now describe colouring construction of covering array on graph G. If G is a k-colourable graph then build a covering array $CA(n,k,g)$ and without loss of generality associate row i of $CA(n,k,g)$ with colour i for $1 \leq i \leq k$. In order to construct $CA(n,G,g)$, we assign row i of $CA(n,k,g)$ to all the vertices having colour i in G.

Recall that if g is prime or power of prime, then one can construct $OA(g+1,g)$. The set of rows in an orthogonal array $OA(k,g)$ is a set of k pairwise qualitatively independent vectors from $\mathbb{Z}_g^{g^2}$. For $g = 2$, by Theorem 2.2.1, there are three qualitatively independent vectors from $\mathbb{Z}_2^4$.

Here we give some examples of graphs where the lower bound in Equation 4.1 is achieved.

Example 4.5.1. If G_1 and G_2 are bicolourable graphs, then $\chi(G_1 \Box G_2) = 2$. Let x_1 and x_2 be two qualitatively independent vectors in $\mathbb{Z}_g^{g^2}$ for $g \geq 2$. Assign vector x_i to all the vertices of $G_1 \Box G_2$ having colour i for $i = 1,2$ to get a covering array with $CAN(G_1 \Box G_2, g) = g^2$.

Example 4.5.2. If G_1 and G_2 are complete graphs, then

$$CAN(G_1 \Box G_2, g) = \max\{CAN(G_1, g), CAN(G_2, g)\}.$$

Example 4.5.3. If G_1 is bicolourable and G_2 is a complete graph on $k \geq 2$ vertices, then $CAN(G_1 \Box G_2, g) = CAN(G_2, g)$. In general, if $\chi(G_1) \leq \chi(G_2)$ and G_2 is a complete graph, then $CAN(G_1 \Box G_2, g) = CAN(G_2, g)$.

Example 4.5.4. Let P_m denote the path of length m and C_n denote the cycle of length n. Then $\chi(P_m \Box C_n) = 3$ when n is odd. Using Theorem 2.2.1, we get a set of three pairwise qualitatively independent vectors in $\mathbb{Z}_g^{g^2}$ for $g \geq 2$. Then the colouring construction of covering arrays gives us a covering array on $P_m \Box C_n$ with $CAN(P_m \Box C_n, g) = g^2$.

Let $*$ represent either the Cartesian, the direct, or the strong product of graphs, and consider a product $G_1 * G_2 * \ldots * G_k$. For any index i, $1 \leq i \leq k$, a *projection map* is defined as:

$$p_i \ : \ G_1 * G_2 * \ldots * G_k \to G_i \text{ where } p_i(x_1, x_2, \ldots, x_k) = x_i.$$

By the definition of the Cartesian, the direct, and the strong product of graphs, each p_i is a weak homomorphism. We now recall the following result from [46] to give another lower bound on $CAN(G_1 \boxtimes G_2, g)$.

Lemma 4.5.1. *Let G_1 and G_2 be graphs and Q be a clique of $G_1 \boxtimes G_2$. Then $Q = p_1(Q) \boxtimes p_2(Q)$, where $p_1(Q)$ and $p_2(Q)$ are cliques of G_1 and G_2, respectively.*

Hence a maximum size clique of $G_1 \boxtimes G_2$ is product of maximum size cliques from G_1 and G_2. That is, $\omega(G_1 \boxtimes G_2) = \omega(G_1)\omega(G_2)$. Using the graph homomorphism, this results into another lower bound on covering array number:

$$CAN(K_{\omega(G_1)\omega(G_2)}, g) \leq CAN(G_1 \boxtimes G_2, g).$$

Following are some examples where this lower bound can be achieved.

Example 4.5.5. If G_1 and G_2 are nontrivial bicolourable graphs then $\omega(G_1 \boxtimes G_2) = \chi(G_1 \boxtimes G_2)$ which is 4. Hence $CAN(G_1 \boxtimes G_2, g) = CAN(K_4, g)$, which is of optimal size.

Example 4.5.6. If G_1 and G_2 are complete graphs, then $G_1 \boxtimes G_2$ is again a complete graph. Hence $CAN(G_1 \boxtimes G_2, g) = CAN(K_{\omega(G_1 \boxtimes G_2)}, g)$.

Example 4.5.7. If G_1 is a bicolourable graph and G_2 is a complete graph on $k \geq 2$ vertices, then $\omega(G_1 \boxtimes G_2) = \chi(G_1 \boxtimes G_2) = 2k$. Hence $CAN(G_1 \boxtimes G_2, g) = CAN(K_{2k}, g)$.

4.6 Optimal size covering arrays over the Cartesian product of graphs

Recall that the set of all automorphisms of a graph G forms a group, denoted $Aut(G)$, the automorphism group of G.

Theorem 4.6.1. *Let G_1 be a graph having the property that $Aut(G_1)$ contains a fixed point free automorphism which maps every vertex to its neighbour. Then for any bicolourable graph G_2,*

$$CAN(G_1 \square G_2, g) = CAN(G_1, g).$$

Proof. Consider the set $\Gamma = \{\phi \in Aut(G_1) \mid \phi(u) \in N(u) \setminus \{u\} \text{ for all } u \in V(G_1)\}$ where $N(u)$ denotes the set of neighbours of u. From the assumption, Γ is not empty. Consider a 2-colouring of G_2 with colours 0 and 1. Define

$$W_0 = \{(u,v) \in V(G_1 \square G_2) \mid \text{colour}(v) = 0\}$$

and

$$W_1 = \{(u,v) \in V(G_1 \Box G_2) \mid \text{colour}(v) = 1\}.$$

Note that W_0 and W_1 partition $V(G_1 \Box G_2)$ into two parts. Let the rows of covering array $CA(G_1,g)$ be indexed by $u_1,u_2,\ldots,u_k$. Form an array C with $|V(G_1 \Box G_2)|$ rows and $CAN(G_1,g)$ columns, indexing rows as (u,v) for $1 \leq u \leq |V(G_1)|$, $1 \leq v \leq |V(G_2)|$. If $(u,v) \in W_0$, row (u,v) is row u of $CA(G_1,g)$; otherwise if $(u,v) \in W_1$, row (u,v) is row $\phi(u)$ of $CA(G_1,g)$. We now verify that C is a $CA(G_1 \Box G_2,g)$. Consider two adjacent vertices (u_1,v_1) and (u_2,v_2) of $G_1 \Box G_2$.

(i) Let (u_1,v_1) and (u_2,v_2) belong to W_i, $i = 0,1$. Then $(u_1,v_1)(u_2,v_2) \in E(G_1 \Box G_2)$ if and only if $u_1u_2 \in E(G_1)$ and $v_1 = v_2$. When (u_1,v_1) and (u_2,v_2) belong to W_0, rows (u_1,v_1) and (u_2,v_2) are rows u_1 and u_2 of $CA(G_1,g)$ respectively. As $u_1u_2 \in E(G_1)$, rows u_1 and u_2 are qualitatively independent in $CA(G_1,g)$. When (u_1,v_1) and (u_2,v_2) belong to W_1, rows (u_1,v_1) and (u_2,v_2) are rows $\phi(u_1)$ and $\phi(u_2)$ of $CA(G_1,g)$ respectively. As $\phi(u_1)\phi(u_2) \in E(G_1)$, rows $\phi(u_1)$ and $\phi(u_2)$ are qualitatively independent in $CA(G_1,g)$. Therefore, rows (u_1,v_1) and (u_2,v_2) are qualitatively independent in C.

(ii) Let $(u_1,v_1) \in W_0$ and $(u_2,v_2) \in W_1$. In this case, $(u_1,v_1)(u_2,v_2)$ is an edge in $G_1 \Box G_2$ if and only if $u_1 = u_2$ and $v_1v_2 \in E(G_2)$. Let $u_1 = u_2 = u$. Rows (u,v_1) and (u,v_2) are rows u and $\phi(u)$ of $CA(G_1,g)$. As ϕ is a fixed point free automorphism that maps every vertex to its neighbour, u and $\phi(u)$ are adjacent in G_1. Therefore, the rows indexed by u and $\phi(u)$ are qualitatively independent in $CA(G_1,g)$; therefore, rows (u_1,v_1) and (u_2,v_2) are qualitatively independent in C. $\square$

Definition 4.6.1. Let H be a finite group and S be a subset of $H \setminus \{id\}$ such that $S = -S$ (i.e., S is closed under inverse). The *Cayley graph* of H generated by S, denoted $Cay(H,S)$, is the undirected graph $G = (V,E)$ where $V = H$ and $E = \{(x,sx) \mid x \in H, s \in S\}$. The Cayley graph is connected if and only if S generates H.

Throughout this chapter by $S = -S$ we mean, S is closed under inverse for a given group operation.

Definition 4.6.2. A *circulant graph* $G(n,S)$ is a Cayley graph on $\mathbb{Z}_n$. That is, it is a graph whose vertices are labelled $\{0,1,\ldots,n-1\}$, with two vertices labelled i and j adjacent if and only if $i-j \pmod n \in S$, where $S \subseteq \mathbb{Z}_n$ with $S=-S$ and $0 \notin S$.

Corollary 4.6.1. *Let $G_1 = G(n,S)$ be a circulant graph and G_2 be a bicolorable graph, then $CAN(G(n,S)\Box G_2, g) = CAN(G(n,S), g)$.*

Proof. Let i and j be two adjacent vertices in G_1. We define a mapping ϕ from $\mathbb{Z}_n$ to $\mathbb{Z}_n$ as follows:

$$\phi(v) = v + j - i \pmod n.$$

It is easy to verify that this function is bijective. To show ϕ is an automorphism, consider a pair of distinct vertices v_1 and v_2 in $G(n,S)$. By definition, v_1v_2 is an edge in $G(n,S)$ if and only if $v_1 - v_2 \pmod n \in S$. Note that,

$$\phi(v_1) - \phi(v_2) = v_1 - v_2 \pmod n.$$

Thus, if v_1v_2 is an edge in $G(n,S)$ then $\phi(v_1)\phi(v_2)$ is an edge in $G(n,S)$. Therefore ϕ is an automorphism of $G(n,S)$. It is easy to verify that $v\phi(v)$ is always an edge. So ϕ sends every vertex v to its neighbour $\phi(v)$. Thus $\phi \in \Gamma$ and the result follows from Theorem 4.6.1. $\square$

For a group H and $S \subseteq H$, we denote conjugation of S by elements of itself as

$$S^S = \{ss's^{-1} \mid s, s' \in S\}.$$

Corollary 4.6.2. *Let H be a finite group and $S \subseteq H \setminus \{id\}$ be a generating set for H such that $S = -S$ and $S^S = S$. Then for $G_1 = Cay(H,S)$ and any bicolorable graph G_2,*

$$CAN(G_1 \Box G_2, g) = CAN(G_1, g).$$

Proof. We will show that there exists a $\phi \in Aut(G_1)$ such that ϕ is stabilizer free. Define $\phi : H \to H$ as $\phi(h) = sh$ for some $s \in S$. It is easy to check that ϕ is bijective and being, $s \neq id$ it is stabilizer free. Now to prove ϕ is a graph homomorphism we need to show it is an adjacency preserving map. It is sufficient to prove that vertices h and $s'h$ are adjacent in

G_1 implies vertices sh and $ss'h$ are adjacent in G_1 for some $s' \in S$. As $ss'h = ss's^{-1}sh$ and $ss's^{-1} \in S$, we have sh and $ss'h$ are adjacent in G_1. Hence $\phi \in \Gamma$ and Theorem 4.6.1 implies the result. $\square$

Example 4.6.1. Let H be an abelian group and $S \subseteq H \setminus \{id\}$ be a generating set for H such that $S = -S$. Then we always get $S^S = S$.

Example 4.6.2. For $H = Q_8 = \{\pm 1, \pm i, \pm j, \pm k\}$ and $S = \{\pm i, \pm j\}$, we have $S^S = S$ and $S = -S$.

Example 4.6.3. For $H = D_8 = \langle a, b | a^2 = 1 = b^4, aba = b^3 \rangle$ and $S = \{ab, ba\}$, we have $S^S = S$ and $S = -S$.

Example 4.6.4. For $H = S_n$ and $S =$ set of all even cycles, we have $S^S = S$ and $S = -S$.

Theorem 4.6.2. *Let H be a finite group and S be a generating set for H such that*

1. *$S = -S$ and $id \notin S$*
2. *$S^S = S$*
3. *there exist s_1 and s_2 in S such that $s_1 \neq s_2$ and $s_1 s_2 \in S$*

then for $G_1 = Cay(H, S)$ and any three colourable graph G_2, we have

$$CAN(G_1 \Box G_2, g) = CAN(G_1, g).$$

Proof. Define three distinct automorphisms of G_1, $\sigma_i : H \to H$, for $i = 0, 1, 2$, as $\sigma_0(u) = u$, $\sigma_1(u) = s_1 u$, $\sigma_2(u) = s_2^{-1} u$. Consider a three colouring of G_2 using the colours $0, 1$ and 2. Let

$$W_i = \big\{(u, v) \in V(G_1 \Box G_2) \mid \text{colour}(v) = i\big\} \text{ for } i = 0, 1, 2.$$

Note that W_0, W_1, and W_2 partition $V(G_1 \Box G_2)$ into three parts. Let the rows of covering array $CA(G_1, g)$ be indexed by $u_1, u_2, \ldots, u_k$. Using $CA(G_1, g)$, form an array C with $|V(G_1 \Box G_2)|$ rows and $CAN(G_1, g)$ columns, indexing rows as (u, v) for $1 \leq u \leq |V(G_1)|$,

$1 \leq v \leq |V(G_2)|$. If $(u,v) \in W_i$, row (u,v) is row $\sigma_i(u)$ of $CA(G_1,g)$. Consider two adjacent vertices (u_1,v_1) and (u_2,v_2) of $G_1 \Box G_2$.

(i) Let (u_1,v_1) and (u_2,v_2) belong to W_i, $i = 0,1,2$. In this case, $(u_1,v_1)(u_2,v_2) \in E(G_1 \Box G_2)$ if and only if $u_1u_2 \in E(G_1)$ and $v_1 = v_2$. When (u_1,v_1) and (u_2,v_2) belong to W_0, rows (u_1,v_1) and (u_2,v_2) are rows u_1 and u_2 of $CA(G_1,g)$. As $u_1u_2 \in E(G_1)$, the rows u_1 and u_2 are qualitatively independent in $CA(G_1,g)$. Let (u_1,v_1) and (u_2,v_2) belong to W_1 (res. W_2). Similarly, as s_1u_1 and s_1u_2 are adjacent in G_1 (res. $s_2^{-1}u_1$ and $s_2^{-1}u_1$ are adjacent in G_1) the rows indexed by s_1u_1 and s_1u_2 (res. $s_2^{-1}u_1$ and $s_2^{-1}u_2$) are qualitatively independent in $CA(G_1,g)$. Hence the rows correspond to vertices (u_1,v_1) and (u_2,v_2) are qualitatively independent in C.

(ii) Let $(u_1,v_1) \in W_i$ and $(u_2,v_2) \in W_j$ for $0 \leq i \neq j \leq 2$. In this case, $(u_1,v_1)(u_2,v_2) \in E(G_1 \Box G_2)$ if and only if $u_1 = u_2$ and $v_1v_2 \in E(G_2)$. Let $u_1 = u_2 = u$.

Let $(u,v_1) \in W_0$ and $(u,v_2) \in W_1$. Then rows (u,v_1) and (u,v_2) are rows u and s_1u of $CA(G_1,g)$ respectively. As u and s_1u are adjacent in G_1, the rows indexed by (u,v_1) and (u,v_2) are qualitatively independent in C.

Let $(u,v_1) \in W_0$ and $(u,v_2) \in W_2$. Then rows (u,v_1) and (u,v_2) are rows u and $s_2^{-1}u$ of $CA(G_1,g)$ respectively. As u and $s_2^{-1}u$ are adjacent in G_1, the rows indexed by (u,v_1) and (u,v_2) are qualitatively independent in C.

Let $(u,v_1) \in W_1$ and $(u,v_2) \in W_2$. Then rows (u,v_1) and (u,v_2) are rows s_1u and $s_2^{-1}u$ of $CA(G_1,g)$ respectively. As $s_1u = s_1s_2s_2^{-1}u$ and $s_1s_2 \in S$, vertices s_1u and $s_2^{-1}u$ are adjacent in G_1. Hence the rows indexed by (u,v_1) and (u,v_2) are qualitatively independent in C. $\Box$

Theorem 4.6.3. *Let H be a finite group and S be a generating set for H such that*

1. *$S = -S$ and $id \notin S$*
2. *$S^S = S$*
3. *there exist s_1 and s_2 in S such that $s_1 \neq s_2$ and $s_1s_2, s_1s_2^{-1} \in S$*

then for $G_1 = Cay(H,S)$ and any four colourable graph G_2, we have

$$CAN(G_1\Box G_2,g)=CAN(G_1,g).$$

Proof. Define four distinct automorphisms of G_1, $\sigma_i : H \to H$, $i=0,1,2,3$ as $\sigma_0(u)=u$, $\sigma_1(u)=s_1u$, $\sigma_2(u)=s_2u$ and $\sigma_3(u)=s_1s_2u$. Consider a four colouring of G_2 using the colours $0,1,2$ and 3. Let

$$W_i=\{(u,v)\in V(G_1\Box G_2) \mid \text{colour}(v)=i\} \text{ for } i=0,1,2,3.$$

Let the rows of covering array $CA(G_1,g)$ be indexed by $u_1,u_2,\ldots,u_k$. Form an array C with $|V(G_1\Box G_2)|$ rows and $CAN(G_1,g)$ columns, indexing rows as (u,v) for $1\le u\le |V(G_1)|$, $1\le v\le |V(G_2)|$. If $(u,v)\in W_i$, row (u,v) is row $\sigma_i(u)$ of $CA(G_1,g)$. Consider two adjacent vertices (u_1,v_1) and (u_2,v_2) of C.

(i) Let (u_1,v_1) and (u_2,v_2) belong to W_i, $i=0,1,2,3$. It is easy to verify that the rows correspond to the vertices (u_1,v_1) and (u_2,v_2) are qualitatively independent.

(ii) Let $(u_1,v_1)\in W_i$ and $(u_2,v_2)\in W_j$ for $0\le i\ne j\le 3$. In this case, $(u_1,v_1)(u_2,v_2)\in E(G_1\Box G_2)$ if and only if $u_1=u_2$ and $v_1v_2\in E(G_2)$. Let $u_1=u_2=u$.

Let $(u,v_1)\in W_0$ and $(u,v_2)\in W_i$ for $i=1,2,3$, then row (u,v_1) and (u,v_2) are rows u and $\sigma_i(u)$ of $CA(G_1,g)$ respectively. Then as u and $\sigma_i(u)$ are adjacent in G_1 the rows correspond to the vertices (u,v_1) and (u,v_2) are qualitatively independent.

Let $(u,v_1)\in W_1$ and $(u,v_2)\in W_2$. Then rows (u,v_1) and (u,v_2) are rows s_1u and s_2u of $CA(G_1,g)$. As $s_1u=s_1s_2^{-1}s_2u$ and $s_1s_2^{-1}\in S$, vertices s_1u and s_2u are adjacent in G_1. Hence the rows correspond to the vertices $(u,v_1)\in W_1$ and $(u,v_2)\in W_2$ are qualitatively independent.

Let $(u,v_1)\in W_1$ and $(u,v_2)\in W_3$. Then rows (u,v_1) and (u,v_2) are rows s_1u and s_1s_2u of $CA(G_1,g)$. As $s_1u=s_1s_2^{-1}s_1^{-1}s_1s_2u$ and $s_1s_2^{-1}s_1^{-1}\in S$ being $S^S=S$, vertices s_1u and s_1s_2u are adjacent in G_1. Hence the rows correspond to the vertices $(u,v_1)\in W_1$ and $(u,v_2)\in W_3$ are qualitatively independent.

Let $(u,v_1)\in W_2$ and $(u,v_2)\in W_3$. Then rows (u,v_1) and (u,v_2) are rows s_2u and s_1s_2u of $CA(G_1,g)$. As $s_2u=s_1^{-1}s_1s_2u$ and $s_1^{-1}\in S$, vertices s_2u and s_1s_2u are adjacent in G_1. Hence the rows correspond to the vertices $(u,v_1)\in W_2$ and $(u,v_2)\in W_3$ are qualitatively independent. $\square$

Example 4.6.5. Let $G = Q_8$ and $S = \{\pm i, \pm j, \pm k\}$. Here $s_1 = i$ and $s_2 = j$.

Example 4.6.6. Let $G = Q_8$ and $S = \{-1, \pm i, \pm j\}$. Here $s_1 = -1$ and $s_2 = i$.

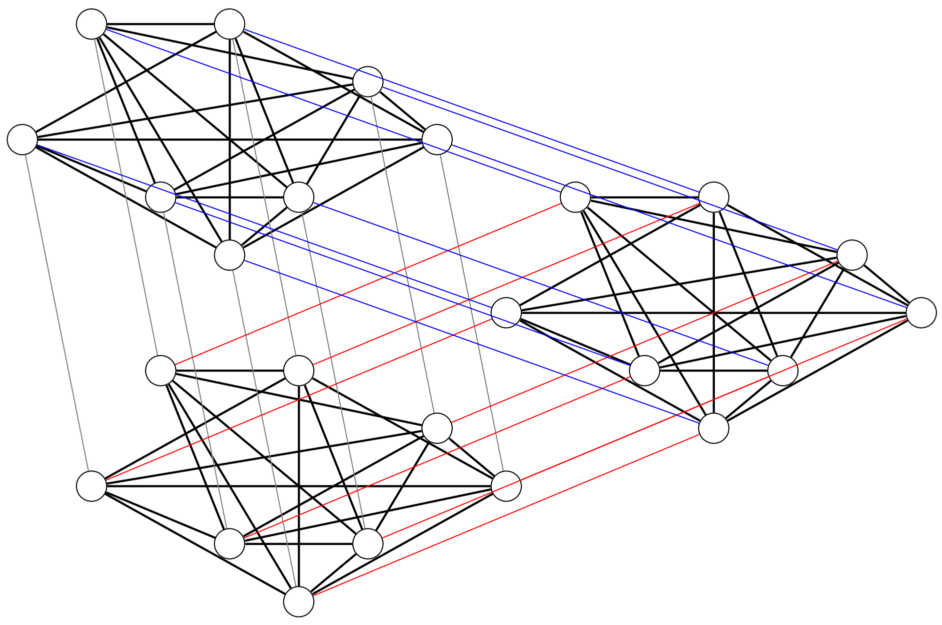


Figure 4.8: $Cay(Q_8, \{-1, \pm i, \pm j\}) \square K_3$

4.7 Approximation algorithm for covering array on graph

In this section, we present an approximation algorithm for construction of a covering array on a given graph $G = (V, E)$ with $k > 1$ prime factors with respect to the Cartesian product. In 1988, G. Seroussi and N H. Bshouty proved that the decision problem whether there exists a binary covering array of strength $t \geq 2$ and size 2^t on a given t-uniform hypergraph is NP-complete [90]. Also, construction of an optimal size covering array on a graph is at least as hard as finding its optimal size. We develop an approximation algorithm to construct covering arrays on graphs with approximation ratio $O(log_s |V|)$, where s can be obtained from the number of symbols corresponding to each vertex. For graphs which are not prime with respect to the Cartesian product, our algorithm improves the best known bounds on $CAN(G,g)$. The following result by Bush is used in our approximation algorithm.

Theorem 4.7.1. [16] *Let g be a positive integer. If g is written in standard form:*

$$g = p_1^{n_1} p_2^{n_2} \dots p_l^{n_l}$$

where $p_1, p_2, \dots, p_l$ are distinct primes, and if

$$r = min(p_1^{n_1}, p_2^{n_2}, \dots, p_l^{n_l}),$$

then one can construct $OA(s,g)$ where $s = 1 + \max(2,r)$.

We are given a wighted connected graph $G = (V,E)$ with each vertex having the same weight g. In our approximation algorithm, we use a technique from [46] for prime factorization of G with respect to the Cartesian product. This can be done in $O(|E| \log |V|)$ time. For details see [46]. After obtaining prime factors of G, we construct strength two covering array C_1 on the maximum size prime factor. Then using the rows of C_1, we produce a covering array on G.

Algorithm 3 APPROX $CA(G,g)$

Input: A weighted connected graph $G=(V,E)$ with $k>1$ prime factors with respect to the Cartesian product. Each vertex has weight g; $g=p_1^{n_1}p_2^{n_2}\dots p_l^{n_l}$ where p_1, $p_2,\dots,p_l$ are primes.

Output: $CA(ug^2,G,g)$.

Step 1: Compute $s=1+\max\{2,r\}$ where $r=\min(p_1^{n_1},p_2^{n_2},\dots,p_l^{n_l})$.

Step 2: Factorize G into k prime factors with respect to the Cartesian product; say $G=\Box_{i=1}^{k}G_i$ where $G_i=(V_i,E_i)$ is a prime factor.

Step 3: Suppose $|V_1|\geq|V_2|\geq\dots\geq|V_k|$. For prime factor $G_1=(V_1,E_1)$ **do**

1. Find the smallest positive integer u such that $s^u\geq|V_1|$. That is, $u=\lceil\log_s|V_1|\rceil$.

2. Let $OA(s,g)$ be an orthogonal array and denote its ith row by R_i for $i=1,2,\dots,s$. Total s^u many row vectors $(R_{i_1},R_{i_2},\dots R_{i_u})$, each of length ug^2, are formed by horizontally concatenating u rows R_{i_1}, R_{i_2}, $\dots$, R_{i_u} where $1\leq i_1,\dots,i_u\leq s$.

3. Form an $|V_1|\times ug^2$ array C_1 by choosing any $|V_1|$ rows out of s^u concatenated row vectors. Let the rows of C_1 be indexed by $0,1,\dots,|V_1|-1$. Each row in the array corresponds to a vertex in the graph G_1.

Step 4: Form an array C with $|V|$ rows and ug^2 columns, indexing rows as $(v_1,v_2,\dots,v_k)$ for $0\leq v_i\leq|V_i|-1$, $i=1,2,\dots,k$. Row $(v_1,v_2,\dots,v_k)$ is row $v_1+v_2+\cdots+v_k \pmod{|V_1|}$ of C_1. Return C.

Theorem 4.7.2. *Algorithm APPROX $CA(G,g)$ is a polynomial-time $\rho(|V|)$ approximation algorithm for covering array on graph problem, where*

$$\rho(|V|)\leq\left\lceil\log_s\frac{|V|}{2^{k-1}}\right\rceil.$$

Proof. **Correctness:** The verification that C is a $CA(ug^2,G,g)$ is straightforward. First, we show that C_1 is a covering array of strength two with $|V_1|$ parameters. Pick any two distinct rows of C_1 and consider the submatrix induced by these two rows. In the submatrix, there must be a column $(R_i, R_j)^T$ where $i \neq j$. Hence each ordered pair of values appears at least once. Now we show that C is a covering array on G. Let $u = (u_1, \ldots, u_k)$ and $v = (v_1, \ldots, v_k)$ be two adjacent vertices in G. It suffices to show that the rows correspond to u and v are qualitatively independent. We know u and v are adjacent in G if and only if $(u_i, v_i) \in E(G_i)$ for exactly one index $1 \leq i \leq k$ and $u_j = v_j$ for $j \neq i$. Hence $u_1 + u_2 + \ldots + u_k \neq v_1 + v_2 + \ldots + v_k \pmod{|V_1|}$ and in Step 4, two distinct rows from C_1 are assigned to the vertices u and v.

Complexity : The time to find s in Step 1 is $O(\ln g)$. The time to factorize graph $G = (V, E)$ in Step 2 is $O(|E| \log |V|)$. In Step 3(1), the smallest positive integer u can be found in $O(\log_s |V_1|)$ time. In Step 3(2), forming one row vector requires $g^2 \log_s |V_1|$ assignments; hence, forming $|V_1|$ row vectors require $O(g^2 |V_1| \log |V_1|)$ time. Thus the total running time of APPROX $CA(G, g)$ is $O(|E| \log |V| + g^2 |V_1| \log_s |V_1| + \ln g)$. Observing that, $g^2 |V_1| \log_s |V_1| \leq g^2 |V| \log_s |V|$, and in practice, $\ln g \leq |E| \log |V|$, we can restate the running time of APPROX $CA(G, g)$ as $O(|E| \log |V| + g^2 |V| \log_s |V|)$.

Approximation ratio: We show that APPROX $CA(G, g)$ returns a covering array that is at most $\rho(|V|)$ times the size of an optimal covering array on G. We know the smallest n for which a $CA(n, G, g)$ may exist is g^2, that is, $CAN(G, g) \geq g^2$. The algorithm returns a covering array on G of size ug^2 where

$$u = \lceil \log_s |V_1| \rceil.$$

As G has k prime factors, the maximum number of vertices in a factor can be $\frac{|V|}{2^{k-1}}$, that is, $|V_1| \leq \frac{|V|}{2^{k-1}}$. Hence

$$u = \lceil \log_s |V_1| \rceil \leq \left\lceil \log_s \frac{|V|}{2^{k-1}} \right\rceil.$$

By relating to the size of the covering array returned to the optimal size, we obtain our approximation ratio

$$\rho(|V|) \leq \left\lceil \log_s \frac{|V|}{2^{k-1}} \right\rceil.$$

□

Chapter 5

Mixed Covering Arrays on 3-Uniform Hypergraphs

Several generalizations of covering arrays have been proposed in order to address different requirements of the testing applications (see Section 1.2). Mixed covering arrays are a generalization of covering arrays that allows different values for different parameters. Covering arrays on graphs are a generalization of covering arrays; in particular, a covering array on a complete graph is a covering array. Meagher, Moura, and Zekaoui studied mixed covering arrays on graphs in detail in [67]. Mixed variable strength covering arrays have been systematically studied in Raaphorst's Ph.D. thesis [77] and also in [23, 24] by Cheng *et al.*

In this chapter, we consider mixed covering arrays on 3-uniform hypergraphs which generalize mixed covering arrays on graphs introduced in [67] but are a special case of mixed variable strength covering arrays introduced in [23, 77]. See also [5]. The motivation for this work is to widen applications of covering arrays to software, hardware, and network testing. In Section 5.1, we outline the necessary background in the theory of hypergraphs. In Section 5.2, we recall the definition of mixed covering arrays on hypergraphs and a lower bound on its minimum size. In Section 5.3, we present results related to balanced and pairwise balanced vectors which are required for basic hypergraph operations. In this section, we prove Conjecture 3.4.27 posted by Raaphorst in [77]. In Section 5.4, we introduce four basic hypergraph operations. Using these operations, we construct optimal mixed covering arrays on α-acyclic 3-uniform hypergraphs, 3-uniform Interval hypergraphs, conformal 3-uniform hypertrees having a binary tree as host tree, 2-tree hypergraphs, and 3-uniform loose cycles. We also give a solution to Conjecture 3.4.28 posted by Raaphorst in [77].

5.1 Hypergraph Theory

In this section, we recall some definitions and relevant results in hypergraph theory from [10, 101].

Definition 5.1.1. A *hypergraph* H is a pair $H = (V,E)$ where $V = \{v_1, v_2, \ldots, v_k\}$ is a set of elements called nodes or vertices, and $E = \{e_1, e_2, \ldots, e_m\}$ is a set of non-empty subsets of V, called hyperedges, such that

$$\bigcup_{i=1}^{m} e_i = V.$$

A *simple hypergraph* is a hypergraph H such that $e_i \subset e_j \Rightarrow i = j$.

Example 5.1.1. An example of a simple hypergraph is shown in Figure 5.1, where the set of vertices is $V = \{v_1, v_2, v_3, v_4, v_5\}$ and the set of edges is $E = \{e_1, e_2, e_3, e_4, e_5\}$, where $e_1 = \{v_1, v_2, v_3\}$, $e_2 = \{v_2, v_3, v_4\}$, $e_3 = \{v_1, v_3, v_4\}$, $e_4 = \{v_1, v_2, v_5\}$, $e_5 = \{v_1, v_3, v_5\}$ and $e_6 = \{v_4, v_5\}$.

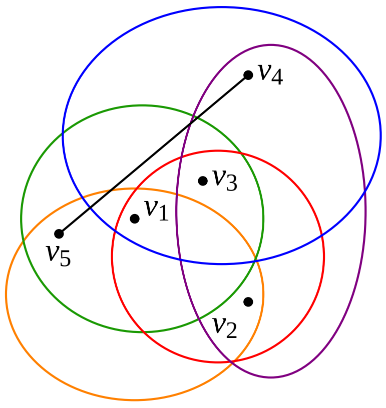


Figure 5.1: A simple hypergraph H.

The *degree* $d_H(v)$ of a vertex $v \in V(H)$ is the number of hyperedges which contain v. The *maximum degree* over all of the vertices in H is denoted by

$$\Delta(H) = \max_{v \in V} d_H(v).$$

A hypergraph is *regular* if all vertices have the same degree. If cardinality of every hyperedge of H is equal to r then H is called r*-uniform* hypergraph. A *complete* r-uniform hypergraph containing k vertices, denoted by K_k^r, is a hypergraph having every r-subset of set of vertices as hyperedge. A partial hypergraph is a hypergraph with some hyperedges removed. More formally, for a set $J \subset \{1, 2, \ldots, m\}$, the *partial hypergraph* generated by J is the hypergraph $(V', \{e_i \mid i \in J\})$ where $V' = \bigcup_{i \in J} e_i$. A subhypergraph is a hypergraph with some vertices removed. Formally, for a set $A \subset V$, the *subhypergraph* H_A induced by A is defined as

$$H_A = \Big(A, \{e_i \cap A \mid \ 1 \leq i \leq m, e_i \cap A \neq \emptyset\}\Big).$$

Example 5.1.2. Consider the hypergraph H shown in Figure 5.1. Let $J = \{2, 4, 6\}$. Then the partial hypergraph generated by J has hyperedges e_2, e_4 and e_6 as shown in Figure 5.2.

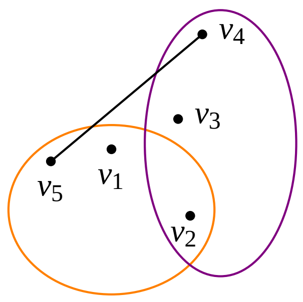


Figure 5.2: A partial hypergraph $H(J)$.

Example 5.1.3. Consider the hypergraph H shown in Figure 5.1. Let $A = \{v_1, v_2, v_4, v_5\}$. Then the subhypergraph H_A induced by A has hyperedges $\{v_1, v_2\}$, $\{v_2, v_4\}$, $\{v_1, v_4\}$, $\{v_1, v_2, v_5\}$, $\{v_1, v_5\}$, and $\{v_4, v_5\}$ as shown in Figure 5.3.

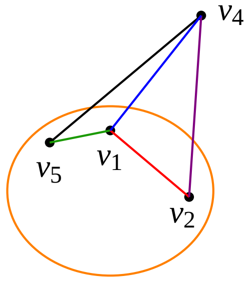


Figure 5.3: A subhypergraph H_A.

5.1.1 Paths and cycles in hypergraphs

In a hypergraph, an alternating sequence

$$v_1, e_1, v_2, e_2, \ldots, v_k, e_k, v_{k+1}$$

of k distinct hyperedges $e_1, e_2, \ldots, e_k$ and $k+1$ distinct vertices $v_1, v_2, \ldots, v_{k+1}$ such that for each $1 \leq i \leq k$, e_i contains v_i and v_{i+1} is called a *path* or *Berge path* connecting vertices v_1 and v_{k+1} or a (v_1, v_{k+1})*-path*. It is called a *cycle* if $v_1 = v_{k+1}$. The value k is called the *length* of the path/cycle respectively. We provide an example of Berge path in Figure 5.4.

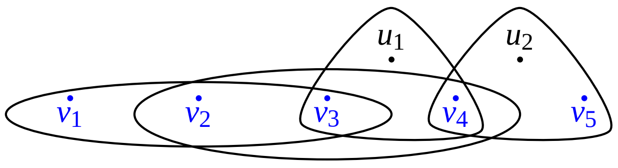


Figure 5.4: A Berge path of length 4 in a 3-uniform hypergraph.

Definition 5.1.2. [87] A *loose cycle* in a hypergraph is a sequence of distinct hyperedges $e_1, e_2, \ldots, e_k$ such that for $1 \leq i \leq k$, $e_i \cap e_{i+1} = \{v_i\}$ where $e_{k+1} = e_1$ and all v_i are distinct; and non-consecutive hyperedges are disjoint. Similarly, a *loose path* in a hypergraph is a sequence of distinct hyperedges $e_1, e_2, \ldots, e_k$ such that for $1 \leq i \leq k-1$, $e_i \cap e_{i+1} = \{v_i\}$ where all v_i are distinct; and non-consecutive hyperedges are disjoint.

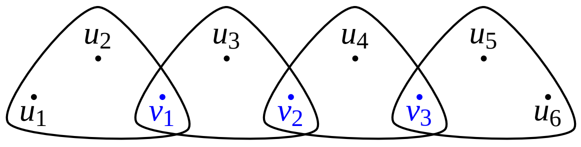


Figure 5.5: A loose path in a 3-uniform hypergraph.

Definition 5.1.3. A *tight cycle* in a t-uniform hypergraph is a sequence of k vertices $v_0, v_1, \ldots,$ v_{k-1} such that $\{v_i, v_{i+1}, \ldots, v_{i+t-1}\}$ is a hyperedge for each i, where i is taken modulo k. Similarly, a *tight path* can also be defined.

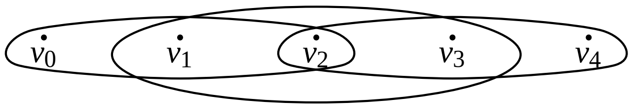


Figure 5.6: A tight path in a 3-uniform hypergraph.

5.1.2 Conformal hypergraphs

The *2-section* of a simple hypergraph H without any loop is the simple graph $[H]_2$ with the same vertices of the hypergraph and edges between all pairs of vertices contained in the same hyperedge.

Example 5.1.4. Consider the hypergraph H in Figure 5.1. It is easy to verify that $[H]_2 = K_5$, a complete graph on five vertices.

Definition 5.1.4. A hypergraph H is *conformal* if all the maximal cliques of the graph $[H]_2$ are hyperedges of H.

Example 5.1.5. Consider the hypergraph $H(J)$ in Figure 5.2. The edges of $[H(J)]_2$ are $\{v_1, v_2\}, \{v_1, v_5\}, \{v_2, v_5\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_2, v_4\}$, and $\{v_4, v_5\}$. The maximal cliques of $[H(J)]_2$ are $\{v_1, v_2, v_5\}, \{v_2, v_3, v_4\}$, and $\{v_4, v_5\}$. All these maximal cliques are hyperedges in $H(J)$. Thus $H(J)$ is a conformal hypergraph. Next, consider the hypergraph H shown in Figure 5.1. Note that $[H]_2$ is a complete graph on five vertices; so the maximal clique size in $[H]_2$ is 5. The hypergraph H does not have any hyperedge containing five vertices. Hence H is not a conformal hypergraph.

5.1.3 Hypertrees

A *host graph* for a hypergraph $H = (V, E)$ is a connected graph on the same vertex set V, such that every hyperedge induces a connected subgraph of the host graph. A host graph which is a tree is called *host tree* of hypergraph H.

Definition 5.1.5. A hypergraph $H = (V, E)$ is called a *hypertree* if there exists a host tree $T = (V, E^{'})$ such that each hyperedge $e_i \in E$ induces a subtree of T.

Example 5.1.6. Let H_1 and H_2 be two hypergraphs as shown in Figure 5.7. We use dotted lines to show the edges of host graphs. Note that H_1 does not have any host tree, hence it is not a hypertree. On the other hand host graph for H_2 is a tree, hence H_2 is a hypertree.

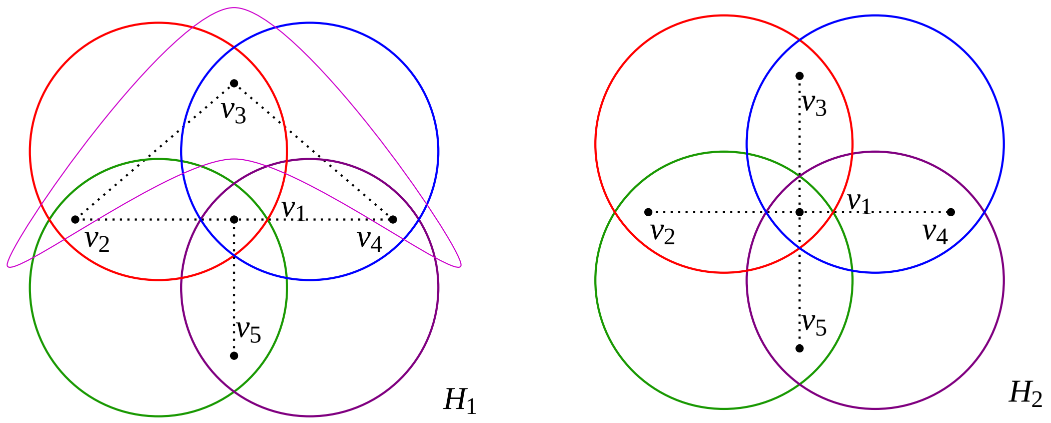


Figure 5.7: A hypergraph H_1 that is not a hypertree and a hypertree H_2.

5.1.4 Hypergraph colourings

There are many generalizations of hypergraph colourings. Here we mention two of the well known generalizations.

Definition 5.1.6. Let $H = (V,E)$ be a hypergraph and $k \geq 2$ be an integer. A *k-colouring* or *weak k-colouring* of the vertices is a partition $(S_1,S_2,...,S_k)$ of the set of vertices into k classes such that every hyperedge which is not a loop meets at least two classes of the partition. In other words, there must be no monochromatic hyperedge with cardinality ≥ 2.

If there exists a k-colouring of vertices then H is said to be *k-colourable*. For a hypergraph H its *chromatic number* $\chi(H)$ is the smallest integer k for which H admits a k-colouring.

Definition 5.1.7. For a hypergraph $H = (V,E)$, a *strong k-colouring* of the vertices is a k-partition $(S_1,S_2,...,S_k)$ of V such that no colour appears more than once in the same hyperedge.

The *strong chromatic number* of a hypergraph H, denoted by $\gamma(H)$, is the smallest integer k for which H admits a strong k-colouring. A strong colouring is always a colouring and hence $\chi(H) \leq \gamma(H)$.

Example 5.1.7. Consider the hypergraph H_2 shown in Figure 5.7. Partition $S_1 = \{v_1\}$ and $S_2 = \{v_2,v_3,v_4,v_5\}$ admits a weak 2-colouring of H_2; and partition $S_1 = \{v_1\}, S_2 = \{v_2,v_4\}$ and $S_3 = \{v_3,v_5\}$ admits a strong 3-colouring of H_2. We also see that $\chi(H_2) = 2$ and $\gamma(H_2) = 3$.

5.2 Mixed covering arrays on hypergraphs

A *weighted hypergraph* is a hypergraph with a positive weight assigned to each vertex. We now recall the definition of mixed covering array on hypergraph from [77].

Definition 5.2.1. Let H be a weighted hypergraph with k vertices and weights $g_1 \leq g_2 \leq \ldots \leq g_k$, and let n be a positive integer. A covering array on H, denoted by $CA(n,H,\prod_{i=1}^{k} g_i)$, is an $k \times n$ array with the following properties:

1. row i corresponds to a vertex $v_i \in V(H)$ with weight g_i;

2. the entries in row i are from $\mathbb{Z}_{g_i}$;

3. if $e = \{v_1, v_2, \ldots, v_t\} \in E(H)$, the rows correspond to vertices $v_1, v_2, \ldots, v_t$ are t-qualitatively independent.

In this chapter we focus on hypergraphs that are 3-uniform, rather than general hypergraphs. Given a weighted hypergraph H with weights $w_H(v_i) = g_i$, $i = 1, 2, \ldots, k$, the *mixed covering array number on H*, denoted by $CAN(H, \prod_{i=1}^{k} g_i)$, is the minimum n for which there exists a $CA(n, H, \prod_{i=1}^{k} g_i)$, that is,

$$CAN\big(H, \prod_{i=1}^{k} g_i\big) = \min_{n \in \mathbb{N}} \Big\{ n \ : \ \exists \text{ a } CA\big(n, H, \prod_{i=1}^{k} g_i\big) \Big\}.$$

Note that

$$CAN\big(H, \prod_{i=1}^{k} w_H(v_i)\big) \geq \max \left\{ \prod_{v_i \in e} w_H(v_i) : \ e \in E(H) \right\} \tag{5.1}$$

A $CA(n, H, \prod_{i=1}^{k} g_i)$ of size $n = CAN(H, \prod_{i=1}^{k} g_i)$ is called *optimal*. A mixed covering array of strength three is a $CA(n, K_k^3, \prod_{i=1}^{k} g_i)$, where K_k^3 is the complete 3-uniform hypergraph on k vertices with weights g_i, for $1 \leq i \leq k$.

Example 5.2.1. Consider the hypergraph H shown in Figure 5.1. Consider a weight function $w : V \to \mathbb{N}$ as $w(v_1) = 3, w(v_2) = 2, w(v_3) = 2, w(v_4) = 2$ and $w(v_5) = 2$. An optimal mixed covering array on H of size 12 is given below.

$$\begin{array}{c} \mathbf{v_1} \\ \mathbf{v_2} \\ \mathbf{v_3} \\ \mathbf{v_4} \\ \mathbf{v_5} \end{array} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$CA(12, H, 3 \cdot 2^4)$

A hypergraph is q-partite if the vertex set V can be partitioned into q sets so that each hyperedge intersects each set at exactly one vertex. Cheng [24] showed that, for a q-partite hypergraph H with maximum degree $t \leq q$, if an $OA(q, g, t)$ exists, then $CAN(H, g) = g^t$.

Cheng *et al.* [23] proved that if H is a hypertree then one can construct optimal mixed covering array on H. However, the hypertrees considered here in Section 5.4 are different from that of in [23]. Raaphorst [77] solved the problem of constructing optimal mixed covering arrays over triangulation hypergraphs of the sphere. He also considered the problem of building mixed covering arrays over 2-trees and gave a conjecture regarding the nature of an optimal construction.

5.3 Balanced and Pairwise Balanced Vectors

In this section, we give several results related to balanced and pairwise balanced vectors which are required for basic hypergraph operations defined in the next section.

Definition 5.3.1. A vector $x \in \mathbb{Z}_g^n$ is *balanced* if for every symbol $a \in \mathbb{Z}_g$, the number of indices i such that $x(i) = a$ is equal to either $\lfloor n/g \rfloor$ or $\lceil n/g \rceil$.

Definition 5.3.2. Two vectors $x_1 \in \mathbb{Z}_{g_1}^n$ and $x_2 \in \mathbb{Z}_{g_2}^n$ are *pairwise balanced* if both vectors are balanced and for every pair of symbols $(a,b) \in \mathbb{Z}_{g_1} \times \mathbb{Z}_{g_2}$, the number of indices i such that $x_1(i) = a$ and $x_2(i) = b$ is equal to either $\lfloor n/g_1g_2 \rfloor$ or $\lceil n/g_1g_2 \rceil$.

Remark 5.3.1. For $n \geq g_1g_2$, pairwise balanced vectors x_1 and x_2 are always qualitatively independent.

In order to prove Theorem 5.3.1, we need the following graph theoretic result. Let G be a graph. A *decomposition* of a graph is a list of subgraphs such that each edge appears in exactly one subgraph in the list. Here by multigraph we mean a graph without loops and multiple edges between vertices are permitted.

Notation: For any positive integer n, the notation $[1,n]$ is used to denote the set $\{1,2,....,n\}$.

Lemma 5.3.1. *Let G be a bipartite multigraph. Assume that the degrees of the vertices in each part differ by no more than 1. Then for any positive integer $h \leq \Delta(G)$, there is a decomposition of the edges in G into h edge-disjoint subgraphs, each with either* $\left\lfloor \frac{|E(G)|}{h} \right\rfloor$

or $\left\lceil \frac{|E(G)|}{h} \right\rceil$ *edges. Moreover, the subgraphs can be chosen so that the degree of each vertex* v *is either* $\left\lfloor \frac{d_G(v)}{h} \right\rfloor$ *or* $\left\lceil \frac{d_G(v)}{h} \right\rceil$.

Proof. We split each vertex $v \in V(G)$ into $\left\lfloor \frac{d_G(v)}{h} \right\rfloor$ vertices of degree h and, if necessary, one vertex of degree $d_G(v) - h\left\lfloor \frac{d_G(v)}{h} \right\rfloor$. This is done by randomly assigning the edges incident to v among these new vertices of degree at most h. Denote this resultant bipartite multigraph by H with maximum degree $\Delta(H) = h$. We know that a bipartite graph H with maximum degree h is the union of h matchings. Thus $E(H)$ is union of h matchings F_0, F_1, $\ldots$, F_{h-1}. Suppose we have two matchings F_0 and F_1 that differ by size more than 1, say F_0 smaller and F_1 larger. Every component of the symmetric difference $F_0 \Delta F_1$ could be an alternating even cycle or an alternating path. Note that it must contain a path, otherwise their sizes are equal. We can find a path component in $F_0 \Delta F_1$ that contains more edges from F_1 than F_0. Swap the F_1 edges with the F_0 edges in this path component. Then the resultant graph has F_0 increased in size by 1 edge, and F_1 decreased in size by 1 edge. Continue this process on F_0, F_1, $\ldots$, F_{h-1} until the sizes are either $\left\lfloor \frac{|E(G)|}{h} \right\rfloor$ or $\left\lceil \frac{|E(G)|}{h} \right\rceil$. Now identify those vertices of H which correspond to the same vertex of G, then F_0, F_1, $\ldots$, F_{h-1} are mapped onto certain edge disjoint subgraphs F'_0, F'_1, $\ldots$, F'_{h-1} of G. Thus each subgraph F'_i contains either $\left\lfloor \frac{|E(G)|}{h} \right\rfloor$ or $\left\lceil \frac{|E(G)|}{h} \right\rceil$ edges.

Next, we prove that in each subgraph F'_i, the degrees of the vertices in each part differ by no more than 1. Since F_i is a matching, there is at most one F_i-edge incident with any of the $\left\lceil \frac{d_G(v)}{h} \right\rceil$ vertices of H correspond to $v \in V$. Hence

$$d_{F'_i}(v) \leq \left\lceil \frac{d_G(v)}{h} \right\rceil.$$

On the other hand, there are $\left\lfloor \frac{d_G(v)}{h} \right\rfloor$ vertices of H correspond to v which have degree h. There must be an F_i-edge starting from each of these, whence

$$d_{F'_i}(v) \geq \left\lfloor \frac{d_G(v)}{h} \right\rfloor.$$

Thus we have $\left\lfloor \frac{d_G(v)}{h} \right\rfloor \leq d_{F'_i}(v) \leq \left\lceil \frac{d_G(v)}{h} \right\rceil$ for $i = 0, 1, \ldots, h-1$. This proves that in each subgraph F'_i, the degrees of the vertices in each part differ by no more than 1. □

Theorem 5.3.1. *Let $x_1 \in \mathbb{Z}_{g_1}^n$ and $x_2 \in \mathbb{Z}_{g_2}^n$ be two balanced vectors. Then for any positive integer $h \leq \max\left\{\left\lceil \frac{n}{g_1} \right\rceil, \left\lceil \frac{n}{g_2} \right\rceil\right\}$, there exists a balanced vector $y \in \mathbb{Z}_h^n$ such that x_1 and y are pairwise balanced and x_2 and y are pairwise balanced.*

Proof. Construct a bipartite multigraph G that corresponds to x_1 and x_2 as follows: G has g_1 vertices in the first part $P \subseteq V(G)$ and g_2 vertices in the second part $Q \subseteq V(G)$. Let $P_a = \{i \mid x_1(i) = a\}$ for $a = 0, 1, \ldots, g_1 - 1$, be the vertices of P, while $Q_b = \{i \mid x_2(i) = b\}$ for $b = 0, 1, \ldots, g_2 - 1$, be the vertices of Q. We have that $\left\lfloor \frac{n}{g_1} \right\rfloor \leq |P_a| \leq \left\lceil \frac{n}{g_1} \right\rceil$ and $\left\lfloor \frac{n}{g_2} \right\rfloor \leq |Q_b| \leq \left\lceil \frac{n}{g_2} \right\rceil$, as x_1 and x_2 are balanced vectors. For each $i = 1, 2, \ldots, n$ there exists exactly one $P_a \in P$ with $i \in P_a$ and exactly one $Q_b \in Q$ with $i \in Q_b$. For each such i, add an edge between vertices corresponding to P_a and Q_b and label it i. Hence $d_G(P_a) = \left\lfloor \frac{n}{g_1} \right\rfloor$ or $\left\lceil \frac{n}{g_1} \right\rceil$ and $d_G(Q_b) = \left\lfloor \frac{n}{g_2} \right\rfloor$ or $\left\lceil \frac{n}{g_2} \right\rceil$. Note that $\Delta(G) = \max\left\{\left\lceil \frac{n}{g_1} \right\rceil, \left\lceil \frac{n}{g_2} \right\rceil\right\}$.

By Lemma 5.3.1, for any positive integer $h \leq \Delta(G)$, there is a decomposition of the edges in G into h edge-disjoint subgraphs $F'_0, F'_1, \ldots, F'_{h-1}$, each with either $\lfloor \frac{n}{h} \rfloor$ or $\lceil \frac{n}{h} \rceil$ edges. These h edge-disjoint subgraphs F'_0, F'_1, $\ldots$, F'_{h-1} of G form a partition of $E(G) = [1, n]$ which we use to build a balanced vector $y \in \mathbb{Z}_h^n$. Each edge disjoint subgraph corresponds to a symbol in $\mathbb{Z}_h$ and each edge corresponds to an index from $[1, n]$. Suppose edge disjoint subgraph F'_c corresponds to symbol $c \in \mathbb{Z}_h$. For each edge i in F'_c, define $y(i) = c$. As each subgraph F'_c contains either $\lfloor \frac{n}{h} \rfloor$ or $\lceil \frac{n}{h} \rceil$ edges, each symbol in $\mathbb{Z}_h$ occurs either $\lfloor \frac{n}{h} \rfloor$ or $\lceil \frac{n}{h} \rceil$ times in y. Hence y is a balanced vector. From Lemma 5.3.1, we have $\left\lfloor \frac{n}{g_1 h} \right\rfloor \leq d_{F'_c}(P_a) \leq \left\lceil \frac{n}{g_1 h} \right\rceil$ for $c = 0, 1, \ldots, h-1$. This means that there exist $\left\lfloor \frac{n}{g_1 h} \right\rfloor$ or $\left\lceil \frac{n}{g_1 h} \right\rceil$ edges $i \in [1, n]$ such that $x_1(i) = a$ and $y(i) = c$. So, x_1 and y are pairwise balanced vectors. Similarly, we can show that y and x_2 are pairwise balanced vectors. $\square$

The following corollary is an easy consequence of Theorem 5.3.1.

Corollary 5.3.1. *Let $x \in \mathbb{Z}_g^n$ be a balanced vector. Then for any positive integer $h \leq \left\lceil \frac{n}{g} \right\rceil$, there exists a balanced vector $y \in \mathbb{Z}_h^n$ such that x and y are pairwise balanced.*

Proof. This follows from Theorem 5.3.1. Set $x_1 = x$ and $x_2 = x$. $\square$

Next, we give a solution to Conjecture 3.4.27 posted by Raaphorst in [77]. In order to prove Theorem 5.3.2, we need the following graph theoretic result.

Lemma 5.3.2. *Let $G = (U \cup V, E)$ be a complete bipartite multigraph. Let $U = \{u_0, u_1, \ldots, u_{g_1-1}\}$ and $V = \{v_0, v_1, \ldots, v_{g_2-1}\}$. Assume that the degrees of the vertices in each part differ by no more than 1 and the number of edges between every pair of vertices $u_a \in U$ and $v_b \in V$ is either $\left\lfloor \frac{|E(G)|}{g_1 g_2} \right\rfloor$ or $\left\lceil \frac{|E(G)|}{g_1 g_2} \right\rceil$. Then, for any h such that $hg_1g_2 \leq |E(G)|$, there is a decomposition of edges in G into h edge-disjoint complete bipartite spanning subgraphs, each with either $\left\lfloor \frac{|E(G)|}{h} \right\rfloor$ or $\left\lceil \frac{|E(G)|}{h} \right\rceil$ edges. Moreover, these subgraphs can be chosen so that the degree of each vertex v is either $\left\lfloor \frac{d_G(v)}{h} \right\rfloor$ or $\left\lceil \frac{d_G(v)}{h} \right\rceil$.*

Proof. It is given that $d_G(u_a) = \left\lfloor \frac{|E(G)|}{g_1} \right\rfloor$ or $\left\lceil \frac{|E(G)|}{g_1} \right\rceil$ for $a = 0, 1, \ldots, g_1 - 1$ and $d_G(v_b) = \left\lfloor \frac{|E(G)|}{g_2} \right\rfloor$ or $\left\lceil \frac{|E(G)|}{g_2} \right\rceil$ for $b = 0, 1, \ldots, g_2 - 1$. We construct a bipartite multigraph H from G as follows: We split each vertex $u_a \in U$ in G into $\left\lfloor \frac{d_G(u_a)}{h} \right\rfloor$ vertices of degree h and, if necessary, one vertex of degree $d_G(u_a) - h\left\lfloor \frac{d_G(u_a)}{h} \right\rfloor$ in H. As $g_2 \leq \left\lfloor \frac{|E(G)|}{hg_1} \right\rfloor$, u_a split into at least g_2 vertices in H from the split operation. Label them u_{a0}, u_{a1}, ..., $u_{a(g_2-1)}$, $u_{ag_2} \cdots$ (g_2 onwards are extra). Similarly, we split each vertex $v_b \in V$ into $\left\lfloor \frac{d_G(v_b)}{h} \right\rfloor$ vertices of degree h and, if necessary, one vertex of degree $d_G(v_b) - h\left\lfloor \frac{d_G(v_b)}{h} \right\rfloor$ in H. Thus, v_b split into at least g_1 vertices in H from the split operation. Label them v_{b0}, v_{b1}, ..., $v_{b(g_1-1)}$, $v_{bg_1} \cdots$ (g_1 onwards are extra). We have at least h edges between every pair of vertices u_a and v_b; consider any h edges between u_a and v_b. These h edges become the h edges between u_{ab} and v_{ba} in H. We add the remaining edges between u_a and v_b arbitrarily to H amongst the extra vertices, provided we maintain H as a bipartite graph with maximum degree h. This results in a graph (possibly multigraph) where every vertex has maximum degree h. We know that a bipartite graph with maximum degree h is the union of h matchings. Thus $E(H)$ is the union of h matchings $F_0, F_1, \ldots, F_{h-1}$. Now identify those vertices of H which correspond to the same vertex of G, then F_0, F_1, $\ldots$, F_{h-1} are mapped onto certain edge disjoint spanning subgraphs F'_0, F'_1, $\ldots$, F'_{h-1} of G. We claim each of the spanning subgraphs F'_i is a complete bipartite multigraph. For every $a \in \mathbb{Z}_{g_1}$, $b \in \mathbb{Z}_{g_2}$, there are h edges between u_{ab} and v_{ba} in H, and they will all appear in different matchings F_0, F_1, $\ldots$, F_{h-1}.

This ensures that the spanning subgraphs contain at least one edge between u_a and v_b for every $a \in \mathbb{Z}_{g_1}$, $b \in \mathbb{Z}_{g_2}$. This proves that each of the spanning subgraphs F_i' is a complete bipartite multigraph.

Since F_i is a matching, there is at most one F_i-edge incident with any of the $\left\lceil \frac{d_G(u_a)}{h} \right\rceil$ vertices of H correspond to $u_a \in U$. Hence

$$d_{F_i'}(u_a) \leq \left\lceil \frac{d_G(u_a)}{h} \right\rceil.$$

On the other hand, there are $\left\lfloor \frac{d_G(u_a)}{h} \right\rfloor$ vertices of H correspond to u_a which have degree h. There must be an F_i-edge starting from each of these, whence

$$d_{F_i'}(u_a) \geq \left\lfloor \frac{d_G(u_a)}{h} \right\rfloor.$$

Thus we have $\left\lfloor \frac{d_G(u_a)}{h} \right\rfloor \leq d_{F_i'}(u_a) \leq \left\lceil \frac{d_G(u_a)}{h} \right\rceil$ for $i = 0, 1, \ldots, h-1$. Similarly, we can show that the degrees of the vertices in V differ by no more than one. Next, we need to show each spanning subgraph F_i' contains either $\left\lfloor \frac{|E(G)|}{h} \right\rfloor$ or $\left\lceil \frac{|E(G)|}{h} \right\rceil$ edges. In other words, this corresponds to each matching F_i contains either $\left\lfloor \frac{|E(G)|}{h} \right\rfloor$ or $\left\lceil \frac{|E(G)|}{h} \right\rceil$ edges. The proof of this part is the same as that of Lemma 5.3.1. □

Theorem 5.3.2. *Let $x_1 \in \mathbb{Z}_{g_1}^n$ and $x_2 \in \mathbb{Z}_{g_2}^n$ be two pairwise balanced vectors. Then for any h such that $g_1 g_2 h \leq n$, there exists a balanced vector $y \in \mathbb{Z}_h^n$ such that x_1, x_2 and y are 3-qualitatively independent and x_1 and y are pairwise balanced and x_2 and y are pairwise balanced.*

Proof. Construct a complete bipartite multigraph $G = (P \cup Q, E)$ that corresponds to x_1 and x_2 as defined in the proof of Theorem 5.3.1. Clearly, $d_G(P_a) = \left\lfloor \frac{n}{g_1} \right\rfloor$ or $\left\lceil \frac{n}{g_1} \right\rceil$ for $a = 0, 1, \ldots, g_1 - 1$ and $d_G(Q_b) = \left\lfloor \frac{n}{g_2} \right\rfloor$ or $\left\lceil \frac{n}{g_2} \right\rceil$ for $b = 0, 1, \ldots, g_2 - 1$. We have that the vectors x_1 and x_2 are pairwise balanced, that is, for each pair $(a, b) \in \mathbb{Z}_{g_1} \times \mathbb{Z}_{g_2}$, the number of edges between P_a and Q_b is $\left\lfloor \frac{n}{g_1 g_2} \right\rfloor$ or $\left\lceil \frac{n}{g_1 g_2} \right\rceil$. From Lemma 5.3.2, for any positive integer h such that $g_1 g_2 h \leq n$, there is a decomposition of edges in G into h edge-disjoint

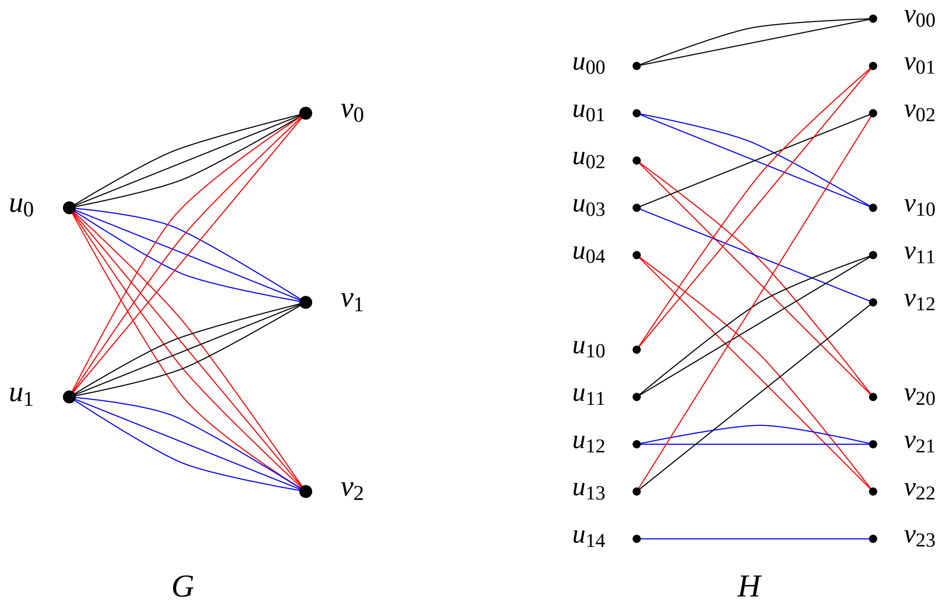


Figure 5.8: An illustration of graphs G and H in the proof of Lemma 5.3.2 with $g_1 = 2, g_2 = 3, |E(G)| = 19$ and $h = 2$.

complete bipartite spanning subgraphs $F'_0, F'_1, \ldots, F'_{h-1}$, each with either $\lfloor \frac{n}{h} \rfloor$ or $\lceil \frac{n}{h} \rceil$ edges; moreover, these subgraphs can be chosen so that the degrees of the vertices in each part differ by no more than 1. These h edge-disjoint spanning subgraphs F'_0, F'_1, $\ldots$, F'_{h-1} of G form a partition of $E(G) = [1,n]$ which we use to build a balanced vector $y \in \mathbb{Z}_h^n$. Each edge-disjoint spanning subgraph corresponds to a symbol in $\mathbb{Z}_h$ and each edge corresponds to an index from $[1,n]$. Suppose edge-disjoint spanning subgraph F'_c corresponds to symbol $c \in \mathbb{Z}_h$. For each edge i in F'_c, define $y(i) = c$. We need to show that x_1, x_2, y are 3-qualitatively independent. For any $a \in \mathbb{Z}_{g_1}$, $b \in \mathbb{Z}_{g_2}$, $c \in \mathbb{Z}_h$, in the spanning subgraph F'_c there is an edge i incident to $P_a \in P$ and $Q_b \in Q$ as F'_c is a complete bipartite multigraph. This means that for any $a \in \mathbb{Z}_{g_1}, b \in \mathbb{Z}_{g_2}, c \in \mathbb{Z}_h$, there exists an $i \in [1,n]$ such that $x_1(i) = a$, $x_2(i) = b$, and $y(i) = c$. So, x_1, x_2 and y are 3-qualitatively independent. Next, we prove that x_1 and y are pairwise balanced, and x_2 and y are pairwise balanced. Since we have $\left\lfloor \frac{n}{g_1 h} \right\rfloor \leq d_{F_c{}'}(P_a) \leq \left\lceil \frac{n}{g_1 h} \right\rceil$ for $c = 0, 1, \ldots, h-1$. This means that there exist $\left\lfloor \frac{n}{g_1 h} \right\rfloor$ or $\left\lceil \frac{n}{g_1 h} \right\rceil$ edges $i \in [1,n]$ such that $x_1(i) = a$ and $y(i) = c$. So, x_1 and y are pairwise balanced vectors.

Similarly, we can show that y and x_2 are pairwise balanced vectors. Next, we need to show that y is balanced. This corresponds to each spanning subgraph F_c' contains either $\lfloor \frac{n}{h} \rfloor$ or $\lceil \frac{n}{h} \rceil$ edges. $\square$

5.4 Optimal mixed covering arrays on hypergraphs

Let H be a weighted 3-uniform hypergraph with k vertices. Label the vertices $v_1, v_2, \ldots, v_k$ and for each vertex v_i denote its associated weight by $w_H(v_i)$.

Definition 5.4.1. The *product weight* of a wighted hypergraph H, denoted $PW(H)$ is defined to be

$$PW(H) = \max \left\{ \prod_{v_i \in e} w_H(v_i) \ : \ e \in E(H) \right\}.$$

Note that $CAN(H, \prod_{i=1}^{k} w_H(v_i)) \geq PW(H)$. Hence a mixed covering array on a hypergraph with size $PW(H)$ is an optimal covering array. If there is no ambiguity about the hypergraph H, we denote $w_H(v)$ by $w(v)$.

Definition 5.4.2. Let H be a weighted hypergraph. A *balanced covering array on H* is a covering array on H in which each row is balanced and the rows correspond to vertices in a hyperedge are pairwise balanced.

5.4.1 Basic hypergraph operations

We introduce four hypergraph operations which will be used to construct optimal size mixed covering arrays on different families of 3-uniform hypergraphs.

1. *Single-vertex edge hooking I*

2. *Single-vertex edge hooking II*

3. *Two-vertex hyperedge hooking*

4. *Single-vertex hyperedge hooking*

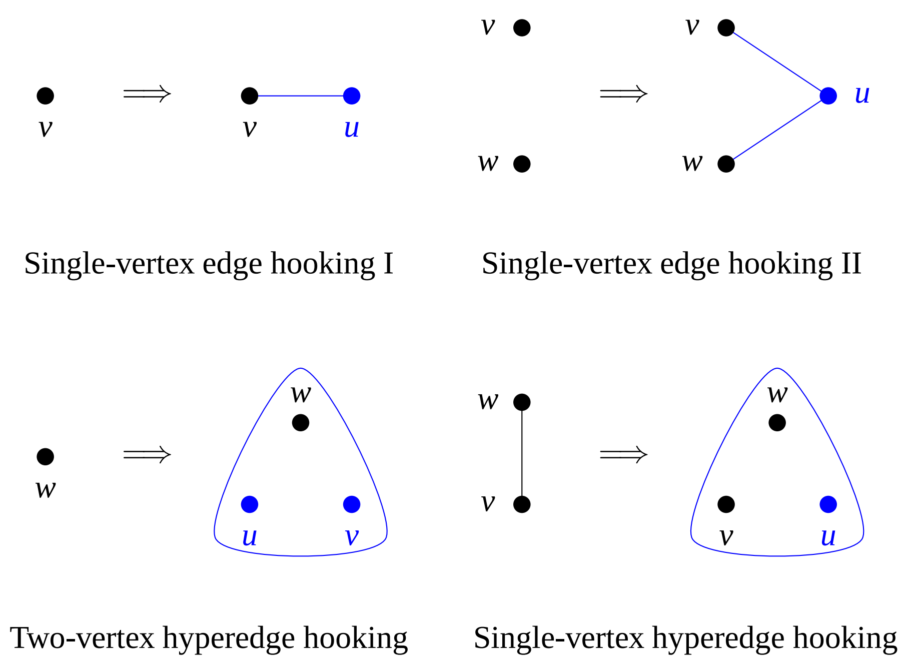

v
u
w
Single-vertex edge hooking I
Single-vertex edge hooking II
Two-vertex hyperedge hooking
Single-vertex hyperedge hooking

Figure 5.9: Basic Hypergraph Operations

A *single-vertex edge hooking I* in a hypergraph H is the operation that inserts a new edge $\{u,v\}$ in which u is a new vertex and v is in $V(H)$. A *single-vertex edge hooking II* in a hypergraph H is the operation that inserts two new edges $\{u,v\}$ and $\{u,w\}$ in which u is a new vertex and v and w are in $V(H)$. A *two-vertex hyperedge hooking* in a hypergraph H is the operation that insert a new hyperedge $\{u,v,w\}$ in which u and v are new vertices and w is in $V(H)$. A *single-vertex hyperedge hooking* in a hypergraph H is the operation that replaces an edge $\{v,w\}$ by a hyperedge $\{u,v,w\}$ where u is a new vertex.

Proposition 5.4.1. *Let H be a weighted hypergraph with k vertices and H' be the weighted hypergraph obtained from H by single-vertex edge hooking I, single-vertex edge hooking II or single-vertex hyperedge hooking operation with u as a new vertex with $w(u)$ such that $PW(H) = PW(H')$. Then, there exists a balanced $CA(n,H,\prod_{i=1}^{k} g_i)$ if and only if there exists a balanced $CA(n,H^{'},w(u)\prod_{i=1}^{k} g_i)$.*

Proof. If there exists a balanced $CA(n,H',w(u)\prod_{i=1}^{k} g_i)$ then by deleting the row corresponding to the new vertex u we can obtain a $CA(n,H,\prod_{i=1}^{k} g_i)$. Conversely, let C_H be a balanced $CA(n,H,\prod_{i=1}^{k} g_i)$. The balanced covering array C_H can be used to construct $C_{H'}$, a balanced $CA(n,H^{'},w(u)\prod_{i=1}^{k} g_i)$. We consider the following cases:

Case 1: Let H' be obtained from H by a single vertex edge hooking I of a new vertex u with a new edge $\{u,v\}$, and $w(u)$ such that $w(u)w(v) \leq PW(H') = PW(H) \leq n$. Using Corollary 5.3.1, we can build a balanced length-n vector **u** over $\mathbb{Z}_{w(u)}$ corresponds to vertex u such that **u** is pairwise balanced with the length-n vector **v** corresponds to vertex v. The array $C_{H'}$ is built by appending row **u** to C_H.

Case 2: Let H' be obtained from H by a single vertex edge hooking II of a new vertex u with two new edges $\{u,v\}$ and $\{u,w\}$, and $w(u)$ such that $w(u)w(v) \leq PW(H') = PW(H) \leq n$ and $w(u)w(w) \leq PW(H') = PW(H) \leq n$. Using Theorem 5.3.1, we can build a balanced length-n vector **u** corresponds to vertex u such that **u** is pairwise balanced with the length-n vectors **v** and **w** correspond to vertices v and w respectively. The array $C_{H'}$ is built by appending row **u** to C_H.

Case 3: Let H' be the graph obtained from H by replacing an edge $\{v,w\} \in E(H)$ by a new hyperedge $\{u,v,w\}$ in which u is a new vertex, and $w(u)$ such that $w(u)w(v)w(w) \leq PW(H') = PW(H) \leq n$. Using Theorem 5.3.2, we can build a balanced length n vector **u** corresponds to vertex u such that **u** is 3-qualitatively independent with two length-n pairwise balanced vectors **v** and **w** correspond to vertices v and w respectively in H. The array $C_{H'}$ is built by appending row **u** to C_H. □

Proposition 5.4.2. *Let H be a weighted hypergraph with k vertices and H' be the weighted hypergraph obtained from H by two-vertex hyperedge hooking operation with u and v as new vertices with $w(u)$ and $w(v)$ such that $PW(H) = PW(H')$. Then, there exists a balanced $CA(n,H,\prod_{i=1}^{k} g_i)$ if and only if there exists a balanced $CA(n,H^{'},w(u)w(v)\prod_{i=1}^{k} g_i)$.*

Proof. If there exists a balanced $CA(n,H',w(u)\prod_{i=1}^{k} g_i)$ then by deleting the rows corresponding to the new vertices u and v we can obtain a $CA(n,H,\prod_{i=1}^{k} g_i)$. Conversely, let C_H be a balanced $CA(n,H,\prod_{i=1}^{k} g_i)$. Hypergraph H' is obtained from H by a two-vertex hyperedge hooking of two new vertices u and v with a new hyperedge $\{u,v,w\}$, and $w(u)$,

$w(v)$ such that $w(u)w(v)w(w) \leq PW(H') = PW(H) \leq n$. Using Corollary 5.3.1, we can build a balanced length-n vector $\mathbf{u}$ corresponds to vertex u such that $\mathbf{u}$ is pairwise balanced with the length-n vector $\mathbf{w}$ corresponds to vertex w. Then using Theorem 5.3.2, we can build a balanced length n vector $\mathbf{v}$ corresponds to vertex v such that $\mathbf{v}$ is 3-qualitatively independent with two length-n pairwise balanced vectors $\mathbf{w}$ and $\mathbf{u}$ correspond to vertices w and u respectively in H. The array $C_{H'}$ is built by appending rows $\mathbf{u}$ and $\mathbf{v}$ to C_H. □

Theorem 5.4.1. *Let H be a weighted hypergraph and $H^{'}$ be a weighted 3-uniform hypergraph obtained from H via a sequence of single-vertex edge hooking I, single-vertex edge hooking II, two-vertex hyperedge hooking, single-vertex hyperedge hooking operations. Let $v_{k+1}, v_{k+2}, ..., v_l$ be the vertices in $V(H^{'}) \setminus V(H)$ with weights $g_{k+1}, g_{k+2}, ..., g_l$ respectively so that $PW(H) = PW(H')$. If there exists a balanced covering array $CA(n, H, \prod_{i=1}^{k} g_i)$, then there exists a balanced $CA(n, H^{'}, \prod_{i=1}^{l} g_i)$.*

Proof. The result is derived by iterating the different cases of Proposition 5.4.1 and Proposition 5.4.2. □

5.4.2 α-acyclic 3-uniform hypergraphs

There are many generalizations of the notion of graph acyclicity in hypergraphs. Graham [45], and independently, Yu and Ozsoyoglu [105], defined α-acyclic property for hypergraphs via a transformation now known as the *GYO reduction*. Given a hypergraph $H = (V, E)$, the GYO reduction applies the following operations repeatedly to H until none of the operations can be applied:

1. If a vertex $v \in V(H)$ has degree one, then delete v from the hyperedge containing it.
2. If $e_1, e_2 \in E(H)$ are distinct hyperedges such that $e_1 \subseteq e_2$, then delete e_1 from $E(H)$.
3. If $e \in E(H)$ is empty, that is $e = \emptyset$, then delete e from $E(H)$.

Definition 5.4.3. A hypergraph H is *α-acyclic* if the GYO reduction on H results in an empty hypergraph.

Example 5.4.1. Consider the hypergraphs $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$ shown in Figure 5.10, where $V_1 = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}, V_2 = \{v_1, v_2, v_3, v_4, v_5, v_6\}$,

$$E_1 = \big\{\{v_1, v_2, v_3\}, \{v_1, v_3, v_4\}, \{v_1, v_2, v_5\}, \{v_2, v_3, v_6\}, \{v_4, v_7, v_8\}\big\} \text{ and}$$

$$E_2 = \big\{\{v_1, v_2, v_3\}, \{v_1, v_3, v_4\}, \{v_2, v_4, v_5\}, \{v_4, v_5, v_6\}\big\}.$$

It is easy to see that H_1 is α-acyclic but H_2 is not α-acyclic.

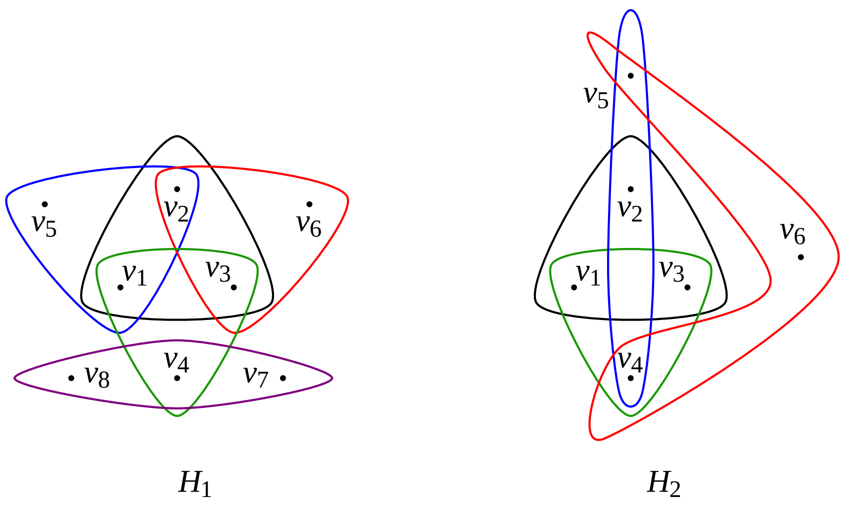


Figure 5.10: An α-acyclic hypergraph H_1 and a non α-acyclic hypergraph H_2.

Theorem 5.4.2. *Let H be a weighted α-acyclic 3-uniform hypergraph with l vertices. Then there exists a balanced mixed $CA(n, H, \prod_{i=1}^{l} g_i)$ with $n = PW(H)$.*

Proof. Apply the GYO reduction on H to record the order in which the hyperedges are deleted. Let $e_1, e_2, \ldots, e_m$ be an ordering in which m hyperedges of H are deleted by the GYO reduction. While constructing covering array on H, consider the hyperedges in reverse order of their deletions. Let H_1 be the hypergraph with the single hyperedge $e_m = \{v_1, v_2, v_3\}$ where $w(v_1) = g_1, w(v_2) = g_2$ and $w(v_3) = g_3$. If $g_1g_2g_3 = n$, construct the $3 \times n$ array A with columns consisting of all triples from $\mathbb{Z}_{g_1} \times \mathbb{Z}_{g_2} \times \mathbb{Z}_{g_3}$. Clearly, A is a balanced covering array on H_1. Otherwise, if $g_1g_2g_3 \leq n$, we construct a balanced covering array of size n on H_1 as follows: begin with a balanced vector $\mathbf{v_1} \in \mathbb{Z}_{g_1}^n$ corresponds to vertex v_1. From Proposition 5.4.2 (using two-vertex hyperedge hooking operation), we get a balanced covering array $CA(n, H_1, \prod_{i=1}^{3} g_i)$. Let H_2 be the hypergraph obtained from H_1 by adding hyperedge e_{m-1}. Using single-vertex hyperedge hooking or two-vertex hyperedge hooking operation, there exists a covering array of size n on H_2. For $i = 2, 3, \ldots, m$, let $H_i = H_{i-1} \cup e_{m+1-i}$. Note that $H_m = H$. As $PW(H_i) \leq PW(H)$ for all $i = 2, 3, \ldots, m$, using single-vertex hyperedge hooking or two-vertex hyperedge hooking operation, there exists a balanced covering array on H_i of size n. In particular, there exists a balanced $CA(n, H, \prod_{i=1}^{l} g_i)$. □

5.4.3 3-uniform interval hypergraphs

A family of hypergraphs called interval hypergraphs is defined in [101]. Here we construct optimal mixed covering arrays on 3-uniform interval hypergraphs.

Definition 5.4.4. A hypergraph $H = (V, E)$ is called an *interval hypergraph* if there exists a linear ordering of the vertices $v_1, v_2, ..., v_n$ such that every hyperedge of H induces an interval in this ordering. In other words, the vertices in V can be placed on the real line such that every hyperedge is an interval.

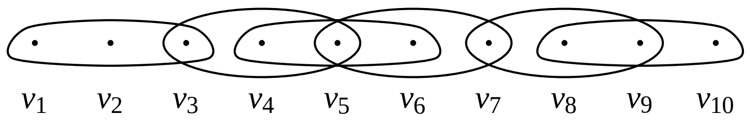


Figure 5.11: A 3-uniform interval hypergraph

Theorem 5.4.3. *Let H be a weighted 3-uniform interval hypergraph with l vertices. Then there exists a balanced mixed $CA(n, H, \prod_{i=1}^{l} g_i)$ where $n = PW(H)$.*

Proof. The result follows immediately from the proof of Theorem 5.4.2 since every interval hypergraph is α-acyclic. □

Corollary 5.4.1. *Let H be a weighted 3-uniform loose path or tight path with l vertices. Then there exists a balanced mixed $CA(n,H,\prod_{i=1}^{l} g_i)$ where $n = PW(H)$.*

Proof. Since every loose path or tight path is an interval hypergraph, the result is an immediate consequence of Theorem 5.4.3. □

5.4.4 3-uniform hypertrees

In this subsection, we give a construction for optimal mixed covering arrays on some specific conformal 3-uniform hypertrees.

Theorem 5.4.4. *Let H be a weighted conformal 3-uniform hypertree with l vertices, having a binary tree as a host tree. Then there exists a balanced mixed $CA(n,H,\prod_{i=1}^{l} g_i)$ with $n = PW(H)$.*

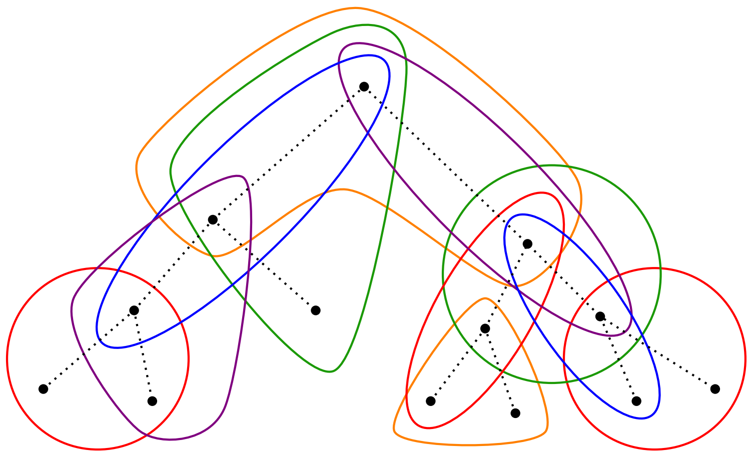


Figure 5.12: A conformal 3-uniform hypertree with a binary host tree

Proof. It suffices to show that H is α-acyclic hypergraph. Let T be a binary host tree of H and let the height of T be h. Since H is a conformal 3-uniform hypergraph, we have $\omega([H]_2) = 3$. Consider the hypergraph F shown in Figure 5.13. F is not conformal since the maximal clique $\{v_1, v_2, v_3, v_4\}$ of its 2-section is not a hyperedge. Thus H contains no partial hypergraph isomorphic to F. If F is a partial hypergraph of H then $4 = \omega([F]_2) \leq \omega([H]_2) = 3$, a contradiction.

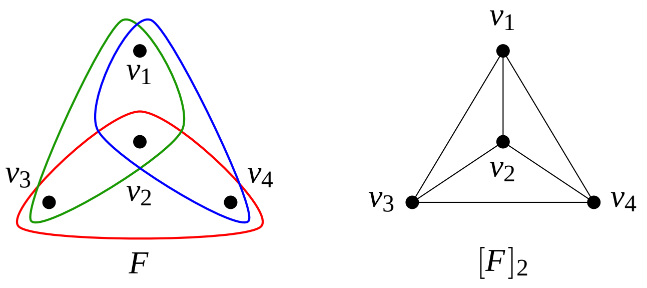


Figure 5.13: A non conformal hypergraph F and its 2-section.

Let u be an internal vertex in T at level $h-1$. Let v and w be its two children at the last level h and p be its parent. Note that at least one of v and w would be of degree 1 in H. If $d_H(v) > 1$ and $d_H(w) > 1$ then the partial hypergraph with hyperedges $\{p,u,v\}$, $\{p,u,w\}$, and $\{u,v,w\}$ is isomorphic to hypergraph F which is a contradiction to conformal property of H. Figure 5.14 shows all possible configurations (up to isomorphism) for the hyperedges in H that contain u. The green coloured hyperedges may or may not be present in H.

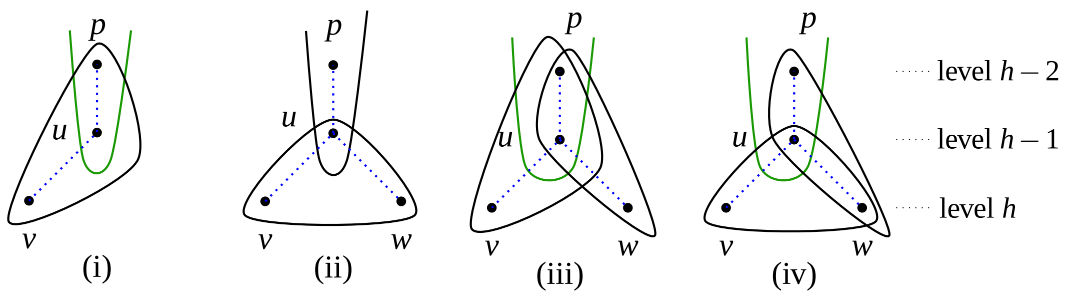


Figure 5.14: All possible configurations for hyperedges that contain u at level $h-1$ in a conformal 3-uniform hypertree with a binary host tree of height h.

Next, we show that one complete iteration of the GYO reduction on H starting at a level h vertex v with $d_H(v) = 1$ results in a conformal 3-uniform partial hypertree with a binary host tree. The GYO reduction on different configurations are shown in Figure 5.15 to Figure 5.18.

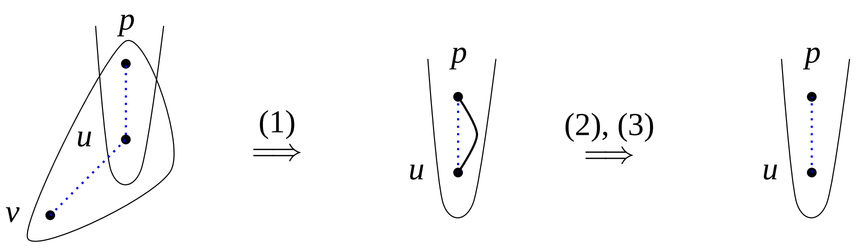


Figure 5.15: One iteration of the GYO reduction on Configuration (i).

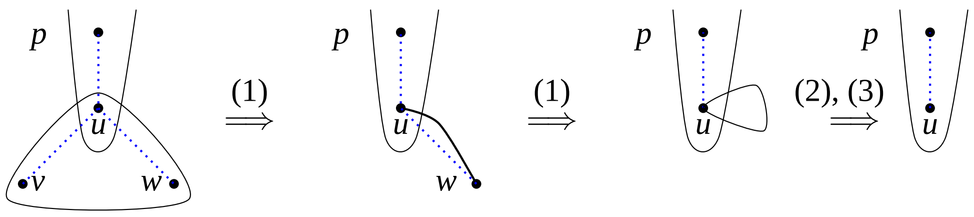


Figure 5.16: One iteration of the GYO reduction on Configuration (ii).

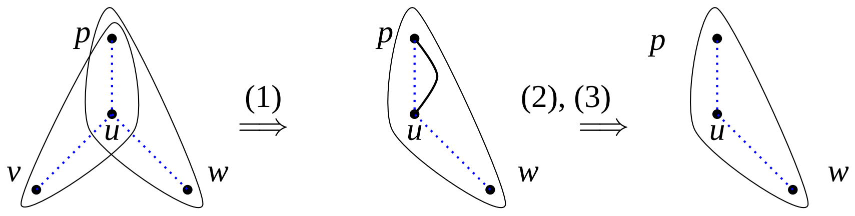


Figure 5.17: One iteration of the GYO reduction on Configuration (iii).

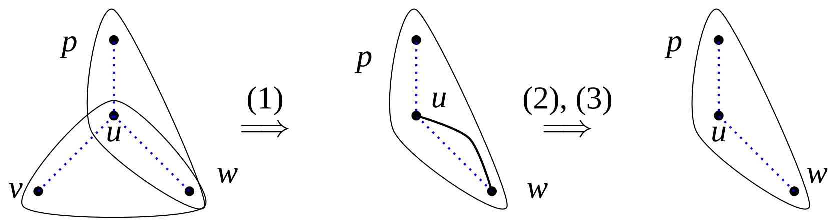


Figure 5.18: One iteration of the GYO reduction on Configuration (iv).

It may be noted that the resultant hypergraph in each case is a conformal 3-uniform hypertree having a binary tree as host tree. Since H has finite number of vertices and hyperedges, a finite number of iterations of the GYO reduction on H result in an empty hypertree. Therefore, H is an α-acyclic hypergraph. Now the proof follows directly from the proof of Theorem 5.4.2. $\square$

5.4.5 2-tree hypergraphs

In this subsection we consider a family of graphs called k-trees, which are generalizations of trees. The following definitions are from [50, 77].

Definition 5.4.5. [50] A graph is a *k-tree* if it can be obtained from K_k by a sequences of vertex additions, where each new vertex is adjacent to a clique of size k in the previously generated graph.

Note that a 1-tree is just a tree. For a k-tree G with $|V(G)| > k$, we can associate a hypergraph with G by replacing each K_{k+1} by a hyperedge of size $k+1$.

Definition 5.4.6. [77] A *k-tree hypergraph* is a hypergraph H constructed as follows:

1. Initially set H to contain precisely the vertices $0, 1, \dots, k$ and hyperedge $\{0, 1, \dots, k\}$.

2. For each new vertex v, select a hyperedge e in H. Pick any vertex $u \in e$ and set $e' = (e \setminus \{u\}) \cup \{v\}$. Then add v and e' to H.

Example 5.4.2. Consider the hypergraph H shown in Figure 5.19, with 7 vertices v_1, v_2, v_3, v_4, v_5, v_6, v_7 and 5 hyperedges $e_1 = \{v_1, v_2, v_3\}$, $e_2 = \{v_1, v_3, v_4\}$, $e_3 = \{v_1, v_2, v_5\}$, $e_4 = \{v_2, v_3, v_6\}$, and $e_5 = \{v_1, v_4, v_7\}$. H is a 2-tree hypergraph as it can be constructed by starting with e_1 and then adding the vertices v_4, v_5, v_6 and v_7 in it using the above described method. It is important to note that there does not exist any acyclic host graph for H, thus H is not a hypertree. On the other hand, the hypergraph H_2 shown in Figure 5.7 is a hypertree but not a 2-tree hypergarph.

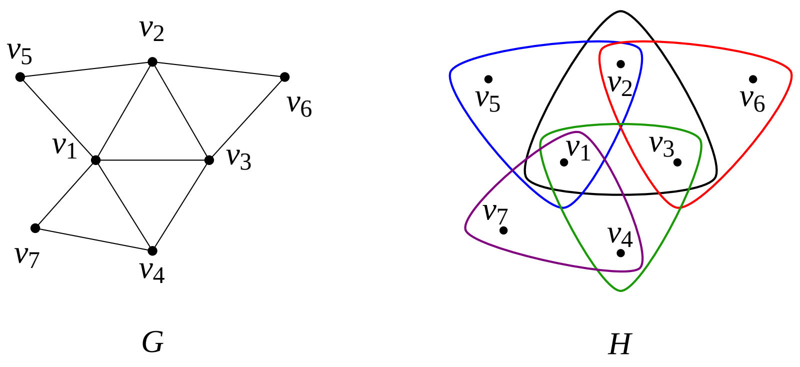


Figure 5.19: A 2-tree graph G and its associated 2-tree hypergraph H.

If H is a k-tree hypergraph, then $[H]_2$ is a k-tree. From Example 5.4.2 it is clear that the hypergraphs considered in this section are different from hypertrees. It is known that, if H is a k-tree hypergraph, then its strong chromatic number $\gamma(H) = k+1$ [77]. The problem of finding a construction for optimal covering arrays over arbitrary, strongly 3-colourable hypergraphs seems to be more difficult problem. The 2-tree hypergraphs are strongly 3-colourable hypergraphs, and given their iterative construction, building optimal arrays over them appears to be considerably more simple. We now consider covering arrays on 2-tree hypergraphs and give a solution to Conjecture 3.4.28 posted by Raaphorst in [77].

Theorem 5.4.5. *Let H be a weighted 2-tree hypergraph with l vertices, and let* $g_1, g_2, \ldots, g_l \geq 1$. *Denote*

$$n = PW(H) = \max\{g_x g_y g_z : \{x,y,z\} \in E(H)\}.$$

Then there exist a balanced mixed $CA(n, H, \prod_{i=1}^{l} g_i)$.

Proof. Begin with a hyperedge $\{x,y,z\} \in E(H)$ such that $g_x g_y g_z = n$. Let H_1 be the hypergraph with the single hyperedge $\{x,y,z\}$. Construct the $3 \times n$ array A with columns consisting of all tuples from $\mathbb{Z}_{g_x} \times \mathbb{Z}_{g_y} \times \mathbb{Z}_{g_z}$. Clearly, A is a balanced covering array on H_1. Let H_2 be a hypergraph obtained from H_1 after inserting a new hyperedge $\{x,y,u\}$ in which u is a new vertex, that is, $H_2 = H_1 \cup \{x,y,u\}$. Let $\mathbf{x}$ and $\mathbf{y}$ be the rows in A correspond to vertices x and y respectively. Using single-vertex hyperedge hooking operation, we can find a balanced row vector $\mathbf{u} \in \mathbb{Z}_{g_u}^n$ correspond to vertex u such that $\mathbf{u}$ and $\mathbf{x}$ are pairwise balanced, **u** and **y** are pairwise balanced and $\mathbf{x}, \mathbf{y}, \mathbf{u}$ are 3-qualitatively independent. Thus, a balanced covering array on H_2 of size n is build by appending row **u** to A. Following the iterative construction for 2-tree hypergraph H, the full $l \times n$ array created in this way is an optimal covering array on H. □

5.4.6 3-uniform loose cycles

The cyclic structure is very rich in hypergraphs as compared to that in graphs [9]. It seems difficult to construct optimal size mixed covering arrays on cycle hypergraphs. There is a special types of 3-uniform cycles for which we are able to construct an optimal size mixed covering arrays.

Theorem 5.4.6. *Let H be a weighted 3-uniform loose cycle $e_1, e_2, \ldots, e_k$ of length k on $2k$ vertices $\{v_1, u_1, v_2, u_2, \ldots, v_k, u_k\}$ such that*

1. *$e_i \cap e_{i+1} = \{v_i\}$ for $i = 1, \ldots, k$ and $e_{k+1} = e_1$;*
2. *$d_H(u_i) = 1$ for each $u_i \in e_i$.*

Let g_i and f_i denote the weights of vertices v_i and u_i respectively. Then there exists a balanced CA$(n, H, \prod_{j=1}^{k} g_j f_j)$ with $n = PW(H)$.

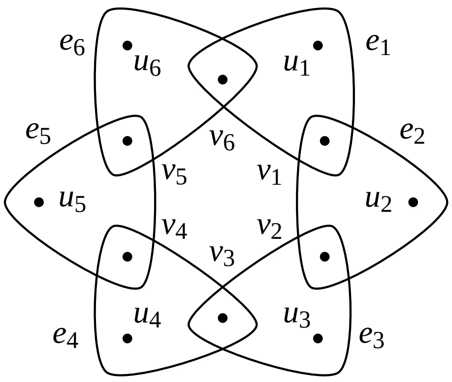


Figure 5.20: A 3-uniform loose cycle of length 6.

Proof. Let $\{v_1,u_2,v_2\}$ be a hyperedge in H with $n = PW(H) = g_1 f_2 g_2$. Let H_1 be the hypergraph with the single hyperedge $\{v_1,u_2,v_2\}$. Construct the $3 \times n$ array with columns consisting of all triples from $\mathbb{Z}_{g_1} \times \mathbb{Z}_{f_2} \times \mathbb{Z}_{g_2}$. Clearly, A is a balanced covering array on H_1. For $i = 2,3,\ldots,k-2$, let H_i be the hypergraph obtained from H_{i-1} after inserting a new edge $\{v_i, v_{i+1}\}$ in which v_{i+1} is a new vertex, that is, $H_i = H_{i-1} \cup \{v_i, v_{i+1}\}$. Using Proposition 5.4.1 (single-vertex edge hooking I operation), for all $i = 2,3,\ldots,k-2$, as $g_i g_{i+1} \leq n$, there exists a balanced $CA(n, H_i, f_2 \prod_{j=1}^{i+1} g_j)$. Let $H_{k-1} = H_{k-2} \cup \{\{v_{k-1}, v_k\}, \{v_k, v_1\}\}$. Using single vertex edge hooking II operation, as $g_{k-1} g_k \leq n$ and $g_1 g_k \leq n$, we get a balanced covering array $CA(n, H_{k-1}, f_2 \prod_{j=1}^{k} g_j)$. Finally, using sequence of single-vertex hyperedge hooking operations on H_{k-1}, replace the edge $\{v_i, v_{i+1}\}$ by the hyperedge $\{v_i, u_{i+1}, v_{i+1}\}$ for $i = 2,3,\ldots,k-1$; also replace the edge $\{v_k, v_1\}$ by the hyperedge $\{v_k, u_1, v_1\}$. Since $g_i f_{i+1} g_{i+1} \leq n$ for all $i = 2,3,\ldots,k-1$ and $g_k f_1 g_1 \leq n$, from Proposition 5.4.1 (using single-vertex hyperedge hooking), there exists a balanced $CA(n, H, \prod_{j=1}^{k} g_j f_j)$. $\square$

Chapter 6

Concluding Remarks and Open Problems

We now give, chapter-wise, the major contributions of the thesis and then list the open problems that arose from this work.

In Chapter 2, we present a construction method for strength four covering arrays, that combines an algebraic technique with the computer search. This construction is an extension of the construction methods developed in [21, 69]. The method proposed here improves many of the best known upper bounds on the sizes of strength four covering arrays for $g = 3$ and $19 \leq k \leq 74$. In the range of k considered here for $g = 3$, the best known results previously come from [30]. In that paper, covering arrays are also found by using a group action on the symbols, but no group action on the rows is employed. Here we expedite and improve the search by also performing a group action on the rows as in [21, 69]. A key advantage of this technique is that we search for either one or two vectors which are used to construct a covering array, rather than searching an entire array. We perform either an exhaustive search or heuristic search to find starter vectors. Is it possible to develop an algorithm that finds the starter vectors directly [66]? Another area to explore is to check other group actions in order to get good covering arrays. We now list the following two questions:

Question 6.1. In Section 2.3, the PGL construction for strength four covering arrays is described. Can this technique be extended to build improved covering arrays of strength five and six?

Question 6.2. Is it possible to modify the PGL construction to get a better upper bound on

4-$CAN(n,g)$ for $g > 3$?

In Chapter 3, an algebraic construction for testing arrays with high 3-way configuration coverage for $g = q+2$ is developed, where q is a prime power. We also present another algebraic construction for testing arrays with high 4-way configuration coverage for $g = q+1$ where q is a prime power. These are useful to create test suites that find a large percentage of errors involving 3- and 4-way interaction of parameters of a system while having a small number of test cases required. A comparison of our constructions with the best known covering array sizes shows that the proposed methods can reduce the number of test cases significantly while compromising only slightly on the coverage. In this chapter, we focus mainly on creating testing arrays with maximum configuration coverage. Depending on software developers requirement, optimization can be with respect to various parameters like simple t-way combination coverage and tuple density. By optimization here we mean maximizing the parameter. Recall from Definition 3.1.2, for a given set of k parameters, simple t-way combination coverage is the proportion of t-way combinations of k parameters for which all parameter-value configurations are fully covered. *Tuple density* of a testing array $\mathscr{A}$ with 100% t-way configuration coverage is the sum of t and the percentage of the covered $(t+1)$-tuples out of all possible $(t+1)$-tuples [22]. The testing array in Example 3.1.1 provides 100% 2-way configuration coverage and 90.625% 3-way configuration coverage; so the tuple density of this testing array is 2.90625. We know a strength t covering array, which by definition covers 100% of t-way parameter-value configurations and has 100% simple t-way combination coverage. One important application of these coverage measures is to get a better understanding of how effective a test suite may be if it is not designed as a covering array. We conclude this section with a question:

Question 6.3. In Section 3.2 and 3.3, two algebraic constructions for testing arrays with high configuration coverage are described. Can these constructions be generalized or extended to build testing arrays with high simple t-way combination coverage for $t \geq 2$ or tuple density?

In Chapter 4, our main contributions are constructions that make optimal covering arrays on large graphs from smaller ones. Large graphs are obtained by considering either the Cartesian, the direct, the strong, or the Lexicographic product of small graphs. One motivation for introducing a graph structure was to optimize covering arrays for their use in testing software and networks based on internal structure. Using graph homomorphisms, we have

$$\max_{i=1,2}\{CAN(G_i,g)\} \leq CAN(G_1\Box G_2,g) \leq CAN(\max_{i=1,2}\{\chi(G_i)\},g) \tag{6.1}$$

We give several classes of Cayley graphs where the lower bound on covering array number $CAN(G_1\Box G_2,g)$ in Equation 6.1 is achievable. It is an interesting problem to find out other classes of graphs for which the lower bound on covering array number $CAN(G_1\Box G_2,g)$ can be achieved. Clearly, another area to explore is to consider in detail the other graph products, that is, the direct, the strong, and the Lexicographic product. We give an approximation algorithm for construction of covering arrays on graphs having $k \geq 1$ factors with respect to the Cartesian product. For graphs having more than one factor, our algorithm improves the present best known bound for the covering array number.

Question 6.4. Suppose G is a prime graph with respect to the Cartesian product but it is factorizable with respect to the strong product. Is it possible to develop a better approximation algorithm to build a covering array on G using graph factorization of G with respect to the strong product?

Question 6.5. There is an algorithm to factorize 3-uniform hypergraphs with respect to the Cartesian product of hypergraphs. Thus we propose the following problem. Is there a polynomial time $\rho(|V|)$-approximation algorithm to construct covering arrays on weighted 3-uniform hypergraphs where each vertex has weight g and $\rho(|V|) = O\left((\log_s |V|)^2\right)$, s is obtained from g?

In Chapter 5, we consider the problem of constructing optimal mixed covering arrays on 3-uniform hypergraphs. Here we extend the results for mixed covering arrays on graphs found in [67]. We introduce four hypergraph operations: single-vertex edge hooking I, single-vertex edge hooking II, two-vertex hyperedge hooking, and single-vertex hyperedge hooking. These operations allow us to add new vertices to a hypergraph, while preserving the size of a balanced covering array on the hypergraph. For the case in which H is a 3-uniform α-acyclic hypergraph, a 3-uniform interval hypergraph, a 3-uniform conformal hypertree with a binary tree as a host tree, a 2-tree hypergraph or a 3-uniform loose cycle, we prove its mixed covering array number is $PW(H)$. We conclude with a list of questions:

Question 6.6. The 2-tree hypergraphs are strongly 3-colourable hypergraphs, and given their iterative construction, we build optimal covering arrays over them in Section 5.4.5. The problem of constructing optimal covering arrays over arbitrary strongly 3-colourable hypergraphs, seems to be more difficult. Identify other families of strongly 3-chromatic hypergraphs $H = (V,E)$, $|V| = k$, such that

$$CAN\big(H, \prod_{i=1}^{k} g_i\big) = PW(H).$$

This question is also raised in [77].

Question 6.7. Find a strongly 3-colourable hypergraph $H = (V,E)$, $|V| = k$, such that

$$CAN\big(H, \prod_{i=1}^{k} g_i\big) > PW(H).$$

Otherwise, show that no such hypergraph exists [77].

Question 6.8. Let H be a weighted 3-uniform tight cycle. Does there exist a covering array $CA(n,H,\prod_{i=1}^{k} g_i)$ with $n = PW(H)$?

Question 6.9. Let $\mathbf{x_1} \in \mathbb{Z}_{g_1}^n$, $\mathbf{x_2} \in \mathbb{Z}_{g_2}^n$ and $\mathbf{x_3} \in \mathbb{Z}_{g_3}^n$ be mutually pairwise balanced and 3-qualitatively independent vectors. Let h be a positive integer so that $h \leq \min\left\{\frac{n}{g_1 g_2}, \frac{n}{g_3}\right\}$. Find a balanced vector $\mathbf{y} \in \mathbb{Z}_h^n$ such that $\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{y}\}$ are 3-qualitatively independent and, $\mathbf{y}$ is pairwise balanced with each $\mathbf{x_i}$ for $i = 1,2,3$. The existence of such a vector $\mathbf{y}$ along with single vertex hyperedge hooking operation will enable us to construct an optimal covering array on the cycle hypergraph H shown in Figure 6.1. Note that H is different from the cycles considered in Section 5.4.6.

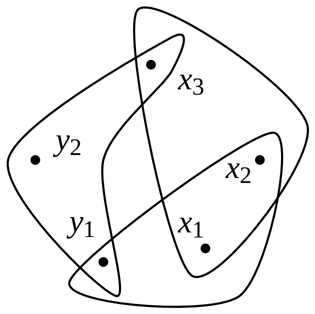


Figure 6.1: A 3-uniform cycle H.

Question 6.10. Let $\mathbf{x_1} \in \mathbb{Z}_{g_1}^n$, $\mathbf{x_2} \in \mathbb{Z}_{g_2}^n$ and $\mathbf{x_3} \in \mathbb{Z}_{g_3}^n$ be mutually pairwise balanced and 3-qualitatively independent vectors. Let h be a positive integer so that $h \leq \min\left\{\frac{n}{g_1 g_2}, \frac{n}{g_2 g_3}\right\}$. Find a balanced vector $\mathbf{y} \in \mathbb{Z}_h^n$ such that $\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{y}\}$ and $\{\mathbf{x_2}, \mathbf{x_3}, \mathbf{y}\}$ are 3-qualitatively independent, and $\mathbf{y}$ is pairwise balanced with each $\mathbf{x_i}$ for $i = 1, 2, 3$. The existence of such a vector $\mathbf{y}$ will enable us to construct an optimal covering array on the hypergraph F shown in Figure 5.13. Thus the conformal condition can be relaxed in Theorem 5.4.4.

Appendix

Sample Programs for strength four covering arrays with $g = 3$

Case 1: Two starter vectors

Program to find the first vector (u)

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

using namespace std;

int h;float m, M;
/*function that counts the number of orbits that are represented
   in each dxy class*/
float rep(int *V,int *n,float m,int k,int x, int y, int z)
{int T[4];
 int A[14];
 for (int i=0;i<k;i++)
   A[i]=0;

  for (int i=0;i<k;i++)
   {  T[0]=V[i];
      T[1]=V[(i+x)%k];
      T[2]=V[(i+x+y)%k];
      T[3]=V[(i+x+y+z)%k];
```

```
if((T[0]==T[1])&&(T[1]==T[2]) &&(T[3]==T[2]))
 A[0]=1;
else if((T[0]==T[1])&&(T[1]==T[2]) && (T[3]!=T[2]))
 A[1]=1;
else if((T[0]!=T[1])&&(T[1]==T[2]) && (T[3]==T[2]))
 A[2]=1;
else if((T[0]!=T[1])&&(T[0]==T[2]) && (T[0]==T[3]))
 A[3]=1;
else if((T[0]==T[1])&&(T[1]!=T[2]) && (T[3]==T[1]))
 A[4]=1;
else if((T[0]==T[1])&&(T[2]==T[3]) && (T[1]!=T[2]))
 A[5]=1;
else if((T[0]==T[2])&&(T[1]==T[3]) && (T[1]!=T[2]))
 A[6]=1;
else if((T[0]==T[3])&&(T[2]==T[1]) && (T[1]!=T[0]))
 A[7]=1;
else if((T[0]==T[1])&&(T[2]!=T[3]) && (T[1]!=T[2]) && (T
    [0]!=T[3]))
 A[8]=1;
else if((T[0]==T[2])&&(T[2]!=T[3]) && (T[1]!=T[3]) && (T
    [1]!=T[2]))
 A[9]=1;
else if((T[0]==T[3])&&(T[0]!=T[1]) && (T[2]!=T[3]) && (T
    [1]!=T[2]))
 A[10]=1;
else if((T[1]==T[3])&&(T[2]!=T[3]) && (T[1]!=T[0]) && (T
    [0]!=T[2]))
 A[11]=1;
else if((T[2]==T[3])&&(T[2]!=T[1]) && (T[2]!=T[0]) && (T
    [0]!=T[1]))
 A[12]=1;
else if((T[2]==T[1])&&(T[1]!=T[0]) && (T[2]!=T[3]) && (T
    [0]!=T[3]))
 A[13]=1;
```

```
      }
      int c=0;

       for (int i=0;i<14;i++)
        { if(A[i]!=1)
            c=c+n[i];
        }
        if(x==z && y==(k-x-y-z))
          {m=m+(k/2)*(pow(3,4)-c);}
        else if(x==y && y==z && z== (k-x-y-z))
         {m=m+(k/4)*(pow(3,4)-c);}
        else
          m=m+k*(pow(3,4)-c);
       return(m);

     }
   /*function that generates dxyz classes*/
   float cov(int *V,int *Vm,int n[14],int k)
    {
     int c,r=0;
     float m;
     m=0;c=0;

     for (int x=1;x<(k/4 +1);x++)
      {for (int y=x;y<k;y++)
        {if(y>(k-2*x)/2.0)
          for(int z=x+1;z<k-2*x-y;z++)
           {m=rep(V,n,m,k,x,y,z);r++;}
         else
          for(int z=x;z<k-2*x-y;z++)
           {m=rep(V,n,m,k,x,y,z); r++;}
        }
      }
    if(k%4==0)
     {m=rep(V,n,m,k,k/4,k/4,k/4);r++;}

     m=m/(((k*(k-1)*(k-2)*(k-3))/24.0)*pow(3,4));
     return(m);
    return(0);

 }

/*function that generates vectors*/
int num(int *V,int *Vm,int n[15],int l,int k,int v)
 {if(m<0 || m>1)
    m=0;
   if(m!=1)
   {int r,s;
   l++;

   for(int i=0;i<v;i++)
   {V[l]=i;
    if(l==k)
    {l++;
     m=cov(V,Vm,n,k);
     if(m>M)
      {M=m;
       cout<<"M="<<M<<"   ";
       for(int j=0;j<k;j++)
         Vm[j]=V[j];
       for(int j=0;j<k;j++)
         cout<<Vm[j];cout<<"\n"<<flush;
      }


    }
    if(l<k)
    r=num(V,Vm,n,l,k,v);
   }
   }
   else if(m==1)
    return(0);
   return(0);

  }

  int main()
  {int k,r,t;
  t=4;
  h=2;
  cout<<"Enter k: ";
  cin>>k;

  int *V,*Vm;
  V = new int[k];
  Vm = new int[k];

  for(int i=0;i<k;i++)
   {V[i]=0;Vm[i]=0;}



  int n[14];
  n[0]=0;
  for(int i=1;i<14;i++)
   n[i]=6;

  r=num(V,Vm,n,-1,k,3);
  return(0);
 }
```

Program that takes the output of the previous program saved in the file conditions.txt and finds the second vector(v)

```
    /*Program that takes the output of the previous program saved in
       the file conditions.txt and finds the second vector(v)*/

#include<iostream>
#include<math.h>
#include<fstream>
#include<sstream>
#include<string>

using namespace std;int w;
/*function that checks whether the passed vector covers all the
   conditions which vector u does not cover*/
int cov(int *V,int *Vm,int *T,int &z,int k)
{z=0;int c;int dx,dy,dz;
 ifstream f;
 f.open("conditions.txt");
 string line;int n;
 while(getline(f,line))
 {istringstream ss(line);
  ss>>dx;ss>>dy;ss>>dz;
  while(ss>>n)
  {c=1;
   for (int i=0;i<k;i++)
   {T[0]=V[i];
    T[1]=V[(i+dx)%k];
    T[2]=V[(i+dx+dy)%k];
    T[3]=V[(i+dx+dy+dz)%k];

    switch(n)
    {case 0:if (c==0) break; else c=0;break;
     case 1:if (c==0) break; else {if((T[0]==T[1])&&(T[1]==T[2])
            && (T[3]!=T[2])) c=0; else c=1;}break;
      case 2:if (c==0) break; else {if((T[0]!=T[1])&&(T[1]==T[2])
            && (T[3]==T[2])) c=0; else c=1;}break;
      case 3:if (c==0) break; else {if((T[0]!=T[1])&&(T[0]==T[2])
            && (T[0]==T[3])) c=0; else c=1;}break;
      case 4:if (c==0) break; else {if((T[0]==T[1])&&(T[1]!=T[2])
            && (T[3]==T[1])) c=0; else c=1;}break;
      case 5:if (c==0) break; else {if((T[0]==T[1])&&(T[2]==T[3])
            && (T[1]!=T[2])) c=0; else c=1;}break;
      case 6:if (c==0) break; else {if((T[0]==T[2])&&(T[1]==T[3])
            && (T[1]!=T[2])) c=0; else c=1;}break;
      case 7:if (c==0) break; else {if((T[0]==T[3])&&(T[2]==T[1])
            && (T[1]!=T[0])) c=0; else c=1;}break;
      case 8:if (c==0) break; else {if((T[0]==T[1])&&(T[2]!=T[3])
            && (T[1]!=T[2]) && (T[0]!=T[3])) c=0; else c=1;}break;
      case 9:if (c==0) break; else {if((T[0]==T[2])&&(T[2]!=T[3])
            && (T[1]!=T[3]) && (T[1]!=T[2])) c=0; else c=1;}break;
      case 10:if (c==0) break; else {if((T[0]==T[3])&&(T[0]!=T[1])
            && (T[2]!=T[3]) && (T[1]!=T[2])) c=0; else c=1;}break;
      case 11:if (c==0) break; else {if((T[1]==T[3])&&(T[2]!=T[3])
            && (T[1]!=T[0]) && (T[0]!=T[2])) c=0; else c=1;}break;
      case 12:if (c==0) break; else {if((T[2]==T[3])&&(T[2]!=T[1])
            && (T[2]!=T[0]) && (T[0]!=T[1])) c=0; else c=1;}break;
      case 13:if (c==0) break; else {if((T[2]==T[1])&&(T[1]!=T[0])
            && (T[2]!=T[3]) && (T[0]!=T[3])) c=0; else c=1;}break;
    }
   }
   if(c==1)
    z++;
   }
  }
  return(z);
 }
```

```
/*function that generates candidate vectors*/
int num(int *V,int *Vm,int *T,int &z,int &Z,int l,int k,int v)
 {if(Z==0)
   return(0);
  else
  {int r,s;
  l++;

  for(int i=0;i<v;i++)
  {V[l]=i;
   if(l==k)
   {l++;
    z=cov(V,Vm,T,z,k);
    if(z<Z)
     {Z=z;
      cout<<"Z="<<Z<<"   ";
      for(int j=0;j<k;j++)
       Vm[j]=V[j];
      for(int j=0;j<k;j++)
       cout<<Vm[j];cout<<"\n"<<flush;
     }


   }
   if(l<k)
   r=num(V,Vm,T,z,Z,l,k,v);
  }
  }
  return(0);
}

 int main()
 {int k,r,t;t=4;
 int z=20000,Z=20000;
```

```
  int T[4];
  for(int i=0;i<4;i++)
   T[i]=0;
  cout<<"Enter k: ";
  cin>>k;

  int *V,*Vm;
  V = new int[k];
  Vm = new int[k];

  for(int i=0;i<k;i++)
   {V[i]=0;Vm[i]=0;}

  r=num(V,Vm,T,z,Z,-1,k,3);

  return(0);
}
```

Case 2: Two vectors u, v and a matrix C_1 and
Case 3: One vector u and a matrix C_1

Program to find the matrix C_1

```
#include<iostream>
#include<math.h>
#include<fstream>
#include<sstream>
#include<string>
#include<time.h>
#include<stdlib.h>

using namespace std;


int cov(int *V[60],int *Vm[60],int *T,int &z,int k,int &s)
{z=0;int c;int x1,x2,x3,x4,x5;
 ifstream f;
 f.open("input.txt");
 string line;int n;
 while(getline(f,line))
 {istringstream ss(line);
  ss>>x1;ss>>x2;ss>>x3;ss>>x4;ss>>x5;

  c=2;
   for(int j=0;j<=s;j++)
   {
    T[0]=V[(x1)%k][j];
    T[1]=V[(x2)%k][j];
    T[2]=V[(x3)%k][j];
    T[3]=V[(x4)%k][j];

   switch(x5)
  {case 0:if (c==0) break; else {if((T[0]==T[1])&&(T[1]==T[2])
      && (T[3]==T[2]) && (T[3]==T[1])) c=0; else c=1;}break;
   case 1:if (c==0) break; else {if((T[0]==T[1])&&(T[1]==T[2])
       && (T[3]!=T[2])) c=0; else c=1;}break;
   case 2:if (c==0) break; else {if((T[0]!=T[1])&&(T[1]==T[2])
       && (T[3]==T[2])) c=0; else c=1;}break;
   case 3:if (c==0) break; else {if((T[0]!=T[1])&&(T[0]==T[2])
       && (T[0]==T[3])) c=0; else c=1;}break;
   case 4:if (c==0) break; else {if((T[0]==T[1])&&(T[1]!=T[2])
       && (T[3]==T[1])) c=0; else c=1;}break;
   case 5:if (c==0) break; else {if((T[0]==T[1])&&(T[2]==T[3])
       && (T[1]!=T[2])) c=0; else c=1;}break;
   case 6:if (c==0) break; else {if((T[0]==T[2])&&(T[1]==T[3])
       && (T[1]!=T[2])) c=0; else c=1;}break;
   case 7:if (c==0) break; else {if((T[0]==T[3])&&(T[2]==T[1])
       && (T[1]!=T[0])) c=0; else c=1;}break;
   case 8:if (c==0) break; else {if((T[0]==T[1])&&(T[2]!=T[3])
       && (T[1]!=T[2]) && (T[0]!=T[3])) c=0; else c=1;}break;
   case 9:if (c==0) break; else {if((T[0]==T[2])&&(T[2]!=T[3])
       && (T[1]!=T[3]) && (T[1]!=T[2])) c=0; else c=1;}break;
   case 60:if (c==0) break; else {if((T[0]==T[3])&&(T[0]!=T[1])
       && (T[2]!=T[3]) && (T[1]!=T[2])) c=0; else c=1;}break;
   case 11:if (c==0) break; else {if((T[1]==T[3])&&(T[2]!=T[3])
       && (T[1]!=T[0]) && (T[0]!=T[2])) c=0; else c=1;}break;
   case 12:if (c==0) break; else {if((T[2]==T[3])&&(T[2]!=T[1])
       && (T[2]!=T[0]) && (T[0]!=T[1])) c=0; else c=1;}break;
   case 13:if (c==0) break; else {if((T[2]==T[1])&&(T[1]!=T[0])
       && (T[2]!=T[3]) && (T[0]!=T[3])) c=0; else c=1;}break;
}
  if(c==0)
  {
   break;}
}
if(c==1)
```

```
    z++;
   }
  f.close();
 return(z);
}

int rand(int *V[60], int *Vm[60], int *T, int k, int &z, int &Z,
   int &s)
{int c;int f;if(Z==0)
  return(0);
 else
  {int r1,r2,r3,p;int n;int zz; string line;
   srand(time(NULL));
   do{p=0;
      for(int j=0;j<k;j++)
       for(int i=0;i<60;i++)
        V[j][i]=0;
   while(Z!=0 && p< 60000)
   {p++;f=1;
    z=cov(V,Vm,T,z,k,s);
   if(z<Z)
   {Z=z;
    cout<<"Z="<<Z<<"\n";
    for(int j=0;j<k;j++)
     for(int i=0;i<60;i++)
     Vm[j][i]=V[j][i];
    for(int j=0;j<k;j++)
     {for(int i=0;i<60;i++)
      cout<<Vm[j][i]<<"  ";
      cout<<"\n";
     }
     remove("output.txt");
     ofstream fo("output.txt");ifstream f;int c,x1,x2,x3,x4,x5;
     f.open("input.txt");
    zz=0;
while(getline(f,line))
{istringstream ss(line);
 ss>>x1;ss>>x2;ss>>x3;ss>>x4;ss>>x5;
 c=2;
  for(int j=0;j<=s;j++)
  {
   T[0]=V[(x1)%k][j];
   T[1]=V[(x2)%k][j];
   T[2]=V[(x3)%k][j];
   T[3]=V[(x4)%k][j];

   switch(x5)
   {case 0:if (c==0) break; else {if((T[0]==T[1])&&(T[1]==T[2])
      && (T[3]==T[2]) &&(T[3]==T[1]) ) c=0; else c=1;}break;
    case 1:if (c==0) break; else {if((T[0]==T[1])&&(T[1]==T[2])
       && (T[3]!=T[2])) c=0; else c=1;}break;
    case 2:if (c==0) break; else {if((T[0]!=T[1])&&(T[1]==T[2])
       && (T[3]==T[2])) c=0; else c=1;}break;
    case 3:if (c==0) break; else {if((T[0]!=T[1])&&(T[0]==T[2])
       && (T[0]==T[3])) c=0; else c=1;}break;
    case 4:if (c==0) break; else {if((T[0]==T[1])&&(T[1]!=T[2])
       && (T[3]==T[1])) c=0; else c=1;}break;
    case 5:if (c==0) break; else {if((T[0]==T[1])&&(T[2]==T[3])
       && (T[1]!=T[2])) c=0; else c=1;}break;
    case 6:if (c==0) break; else {if((T[0]==T[2])&&(T[1]==T[3])
       && (T[1]!=T[2])) c=0; else c=1;}break;
    case 7:if (c==0) break; else {if((T[0]==T[3])&&(T[2]==T[1])
       && (T[1]!=T[0])) c=0; else c=1;}break;
    case 8:if (c==0) break; else {if((T[0]==T[1])&&(T[2]!=T[3])
       && (T[1]!=T[2]) && (T[0]!=T[3])) c=0; else c=1;}break;
    case 9:if (c==0) break; else {if((T[0]==T[2])&&(T[2]!=T[3])
       && (T[1]!=T[3]) && (T[1]!=T[2])) c=0; else c=1;}break;
    case 60:if (c==0) break; else {if((T[0]==T[3])&&(T[0]!=T[1])
          && (T[2]!=T[3]) && (T[1]!=T[2])) c=0; else c=1;}break;
      case 11:if (c==0) break; else {if((T[1]==T[3])&&(T[2]!=T[3])
          && (T[1]!=T[0]) && (T[0]!=T[2])) c=0; else c=1;}break;
      case 12:if (c==0) break; else {if((T[2]==T[3])&&(T[2]!=T[1])
          && (T[2]!=T[0]) && (T[0]!=T[1])) c=0; else c=1;}break;
      case 13:if (c==0) break; else {if((T[2]==T[1])&&(T[1]!=T[0])
          && (T[2]!=T[3]) && (T[0]!=T[3])) c=0; else c=1;}break;
    }

    if(c==0)
      {break;}
    }
    if(c==1)
     {zz++;
      fo<<(x1)%k<<" "<<(x2)%k<<" "<<(x3)%k<<" "<<x4<<"  "<<x5<<" "
          <<endl;
     }

  }f.close();fo.close();
    }
    else
    {for(int j=0;j<k;j++)
      for(int i=0;i<60;i++)
       V[j][i]=Vm[j][i];
    }
    for(int i=0;i<6;i++)
     V[rand()%k][s]=rand()%3;

  if(p == 60000)
    {for(int j=0;j<k;j++)
      for(int i=0;i<60;i++)
       V[j][i]=Vm[j][i];
    }
}
```

```
    }while(f==1);
    }
     return(0);
  }

  int main()
  {int k;
   int r;
   int z,Z; z=20000;Z=20000;
   cout<<"Enter k:";cin>>k;

   int T[4];
   for(int i=0;i<4;i++)
    T[i]=0;

   int **V=new int *[60];
   int **Vm=new int*[60];
   for(int i=0;i<60; ++i)
   {V[i] = new int[k];
   }
   for(int i=0;i<k;i++)
    for(int j=0;j<60;j++)
     {V[i][j]=0;Vm[i][j]=0;}

   for(int s=0;s<60;s++)
    {r=rand(V,Vm,T,k,z,Z,s);
     remove("input.txt");

     ifstream fi("output.txt");
     ofstream fo("input.txt");
     string l;int x1,x2,x3,x4,x5;
    while(getline(fi,l))
    {istringstream ss(l);
     ss>>x1;ss>>x2;ss>>x3;ss>>x4;ss>>x5;
   fo<<x1<<" "<<x2<<" "<<x3<<"  "<<x4<<" "<<x5<<endl;
  }
  fi.close();
  fo.close();
 }

 return(0);
}
```

Bibliography

[1] B. Agarwal, S. Tayal, and M. Gupta. *Softrware Engineering and Testing: An Introduction*. Jones and Bartlett, Sudbury, Massachusetts, 2010.

[2] B. S. Ahmed and K. Z. Zamli. A review of covering arrays and their application to software testing. *Journal of Computer Science*, 7(9):1375–1385, 2011.

[3] Y. Akhtar and S. Maity. Covering arrays on product graphs. In *47th Southeastern International Conference on Combinatorics, Graph theory and Computing*. Boca Raton, FL. March 2016.

[4] Y. Akhtar and S. Maity. Mixed covering arrays on hypergraphs. In *ICECCS*, Communications in Computer and Information Science, pages 327–338. Springer Berlin Heidelberg, 2012.

[5] Y. Akhtar and S. Maity. Mixed covering arrays on 3-uniform hypergraphs. *Preprint*, 2016.

[6] Y. Akhtar, S. Maity, and R. C. Chandrasekharan. Covering arrays of strength four and software testing. In *Mathematics and Computing: ICMC*, Springer Proceedings in Mathematics and Statistics, pages 391–398. Springer India, 2015.

[7] Y. Akhtar, S. Maity, and R. C. Chandrasekharan. Test suites with high 3-way configuration coverage. *Preprint*, 2016.

[8] Y. Akhtar, S. Maity, R. C. Chandrasekharan, and C. J. Colbourn. Improved strength four covering arrays with three symbols. *Preprint*, 2016.

[9] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[10] C. Berge. *Hypergraphs-Combinatorics of Finite Sets*. Elsevier Science, 1989.

[11] S. Y. Borodai and I. S. Grunskii. Recursive generation of locally complete tests. *Cybernetics and Systems Analysis*, 28(4):504–508, 1992.

[12] R. C. Bose and B. Manvel. *Introduction to Combinatorial Theory*. John Wiley & Sons, Inc., New York, NY, USA, 1984.

[13] R. C. Bose, S. S. Shrikhande, and E. T. Parker. Further results on the construction of mutually orthogonal Latin squares and the falsity of Euler's conjecture. *Canad. J. Math.*, 12:189–203, 1960.

[14] R. C. Bryce and C. J. Colbourn. The density algorithm for pairwise interaction testing. *Software Testing, Verification and Reliability*, 17(3):159–182, 2007.

[15] R. C. Bryce and C. J. Colbourn. A density-based greedy algorithm for higher strength covering arrays. *Software Testing, Verification and Reliability*, 19(1):37–53, 2009.

[16] K. A. Bush. A generalization of a theorem due to MacNeish. *Ann. Math. Statist.*, 23(2):293–295, 1952.

[17] K. A. Bush. Orthogonal arrays of index unity. *The Annals of Mathematical Statistics*, 23(3):426–434, 1952.

[18] C. T. Carroll. The cost of poor testing: A U.S. government study (part 1). *EDPACS*, 31(1):1–17, 2003.

[19] J. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143 – 154, 1979.

[20] M. Chateauneuf and D. L. Kreher. On the state of strength three covering arrays. *Journal of Combinatorial Designs*, 10(4):217–238, 2002.

[21] M. A. Chateauneuf, C. J. Colbourn, and D. L. Kreher. Covering arrays of strength three. *Designs, Codes and Cryptography*, 16(3):235–242, 1999.

[22] B. Chen and J. Zhang. Tuple density: A new metric for combinatorial test suites (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, pages 876–879, Waikiki, Honolulu, HI, USA, 2011. ACM.

[23] C. Cheng, A. Dumitrescu, and P. Schroeder. Generating small combinatorial test suites to cover input-output relationships. In *Proceedings of the Third International Conference on Quality Software*, pages 76–82, Washington, DC, USA, 2003. IEEE Computer Society.

[24] C. T. Cheng. The test suite generation problem: Optimal instances and their implications. *Discrete Applied Mathematics*, 155(15):1943–1957, 2007.

[25] S. Chowla, P. Erdős, and E. G. Straus. On the maximal number of pairwise orthogonal Latin squares of a given order. *Canad. J. Math.*, 12:204–208, 1960.

[26] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: an approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, 1997.

[27] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton. The combinatorial design approach to automatic test generation. *IEEE Softw.*, 13(5):83–88, Sept. 1996.

[28] G. D. Cohen and G. Zémor. Intersecting codes and independent families. *IEEE Trans. Information Theory*, 40(6):1872–1881, 1994.

[29] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, C. J. Colbourn, and J. S. Collofello. Variable strength interaction testing of components. In *Proceedings of the 27th An-*

nual International Conference on Computer Software and Applications, COMPSAC, pages 413–418, Washington, DC, USA, 2003. IEEE Computer Society.

[30] C. Colbourn. Conditional expectation algorithms for covering arrays. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 90:97–115, 2014.

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

[32] C. J. Colbourn. Combinatorial aspects of covering arrays. *Le Matematiche*, 59:125–172, 2004.

[33] C. J. Colbourn and J. H. Dinitz. *Handbook of Combinatorial Design*. CRC Press, 1996.

[34] C. J. Colbourn, S. S. Martirosyan, G. L. Mullen, D. Shasha, G. B. Sherwood, J. L. Yucas, and S. S. Martirosyan. Products of mixed covering arrays of strength two. *Journal of Combinatorial Designs*, 14:124–138, 2006.

[35] C. J. Colbourn, C. Shi, C. Wang, and J. Yan. Mixed covering arrays of strength three with few factors. *Journal of Statistical Planning and Inference*, 141(11):3640–3647, 2011.

[36] P. Danziger, E. Mendelsohn, L. Moura, and B. Stevens. Covering arrays avoiding forbidden edges. *Theor. Comput. Sci.*, 410(52):5403–5414, Dec. 2009.

[37] A. Dey and R. Mukherjee. *Fractional Factorial Plans*. John Wiley and Sons, New York, 1999.

[38] T. Dutta and B. K. Roy. Construction of symmetric balanced squares. *Ars Combinatoria*, 47:49–64, 1997.

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

[40] P. Erdös, C. Ko, and R. Rado. Intersection theorems for systems of finite sets. *Quart. J. Math. Oxford*, 12(2):313–320, 1961.

[41] N. Francetić. *Covering arrays with row limit*. PhD thesis, University of Toronto, 2012.

[42] N. Francetić, P. Danziger, and E. Mendelsohn. Group divisible covering designs with block size 4: A type of covering array with row limit. *Journal of Combinatorial Designs*, 21(8):311–341, 2013.

[43] L. Gargano, J. Körner, and U. Vaccaro. Sperner capacities. *Graphs and Combinatorics*, 9(1):31–46, 1993.

[44] A. P. Godbole, D. E. Skipper, and R. A. Sunley. t-covering arrays: Upper bounds and Poisson approximations. *Combinatorics, Probability and Computing*, 5:105–117, 1996.

[45] M. Graham. On the universal relation. Report, University of Toronto, Toronto, Ontario, Canada, 1979.

[46] R. Hammack, W. Imrich, and S. Klavžar. *Handbook of Product Graphs*. CRC Press, 2011.

[47] A. Hartman. Software and hardware testing using combinatorial covering suites. In *Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications*, pages 237–266, Boston, MA, 2005. Springer US.

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

[49] A. Hedayat, N. Sloane, and J. Stufken. *Orthogonal Arrays: Theory and Applications*. Springer series in statistics. Springer-Verlag New York Inc., 1999.

[50] P. Hell and J. Nešetřil. *Graphs and Homomorphisms*. Oxford University Press, Oxford, 2004.

[51] J. D. Horton, B. K. Roy, P. J. Schellenberg, and D. R. Stinson. On decomposing graphs into isomorphic uniform 2-factors. *Annals of Discrete Mathematics (27): Cycles in Graphs*, 115:297 – 319, 1985.

[52] D. S. Hoskins, C. J. Colbourn, and D. C. Montgomery. Software performance testing using covering arrays: Efficient screening designs with categorical factors. In *Proceedings of the 5th International Workshop on Software and Performance*, pages 131–136, New York, USA, 2005. ACM.

[53] C. Kaner, J. Falk, and H. Q. Nguyen. *Testing Computer Software, 2nd Ed. New York*. John Wiley and Sons, Inc., 1999.

[54] G. O. H. Katona. Two applications (for search theory and truth functions) of Sperner type theorems. *Periodica Mathematica Hungarica*, 3(1):19–26, 1973.

[55] D. J. Kleitman and J. Spencer. Families of k-independent sets. *Discrete Mathematics*, 6(3):255–262, 1973.

[56] D. R. Kuhn, R. N. Kacker, and Y. Lei. *Introduction to Combinatorial Testing*. CRC Press, Boca Raton, Florida, 2013.

[57] D. R. Kuhn, R. N. Kacker, and Y. Lei. Measuring and specifying combinatorial coverage of test input configurations. *Innovations in Systems and Software Engineering*, pages 1–13, 2015.

[58] D. R. Kuhn, D. R. Wallace, and A. M. Gallo, Jr. Software fault interactions and implications for software testing. *IEEE Trans. Softw. Eng.*, 30(6):418–421, 2004.

[59] Y. Lei and K. C. Tai. In-parameter-order: A test generation strategy for pairwise testing. In *HASE*, pages 254–261. IEEE Computer Society, 1998.

[60] J. Lobb, C. Colbourn, P. Danziger, B. Stevens, and J. Torres-Jimenez. Cover starters for covering arrays of strength two. *Discrete Mathematics*, 312(5):943–956, 3 2012.

[61] S. Maity. 3-way software testing with budget constraints. *IEICE Transactions*, 95-D(9):2227–2231, 2012.

[62] S. Maity. Software testing with budget constraints. In *Ninth International Conference on Information Technology: New Generations, ITNG, Las Vegas, Nevada, USA*, pages 258–262, 2012.

[63] S. Maity and A. Nayak. Improved test generation algorithms for pair-wise testing. In *IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 235–244. IEEE Computer Society, 2005.

[64] S. Martirosyan and V. T. Trung. On t-covering arrays. *Designs, Codes and Cryptography*, 32(1):323–339, 2004.

[65] C. Maughan. *Test Case Generation Using Combinatorial Based Coverage for Rich Web Applications*. PhD thesis, Logan, UT: Utah State University, 2012.

[66] K. Meagher. *Covering arrays on graphs: qualitative independence graphs and extremal set partition theory*. PhD thesis, University of Ottawa, 2008.

[67] K. Meagher, L. Moura, and L. Zekaoui. Mixed covering arrays on graphs. *Journal of Combinatorial Designs*, 15(5):393–404, 2007.

[68] K. Meagher and B. Stevens. Covering arrays on graphs. *Journal of Combinatorial Theory, Series B*, 95(1):134–151, 2005.

[69] K. Meagher and B. Stevens. Group construction of covering arrays. *Journal of Combinatorial Designs*, 13(1):70–77, 2005.

[70] A. M. Memon and Q. Xie. Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software. *IEEE Transactions on Software Engineering*, 31(10):884–896, 2005.

[71] C. Montanez, D. Kuhn, M. Brady, R. Rivello, J. Reyes, and M. Powers. Evaluation of fault detection effectiveness for combinatorial and exhaustive selection of discretized test inputs. Technical report, Software Quality Professional- June, 2012.

[72] L. Moura, J. Stardom, B. Stevens, and A. Williams. Covering arrays with mixed alphabet sizes. *Journal of Combinatorial Designs*, 11(6):413–432, 2003.

[73] P. Nayeri, C. J. Colbourn, and G. Konjevod. Randomized post-optimization of covering arrays. *Eur. J. Comb.*, 34(1):91–103, 2013.

[74] K. J. Nurmela. Upper bounds for covering arrays by tabu search. *Discrete Applied Mathematics*, 138:143–152, 2004. Optimal Discrete Structures and Algorithms.

[75] P. R. J. Östergård. Constructions of mixed covering codes. Research Report A18, Helsinki University of Technology, Department of Computer Science and Engineering, Digital Systems Laboratory, Espoo, Finland, December 1991.

[76] S. Poljak and Z. Tuza. On the maximum number of qualitatively independent partitions. *Journal of Combinatorial Theory, Series A*, 51(1):111–116, 1989.

[77] S. Raaphorst. *Variable strength covering arrays*. PhD thesis, University of Ottawa, 2013.

[78] C. R. Rao. Factorial experiments derivable from combinatorial arrangements of arrays. *Supplement to the Journal of the Royal Statistical Society*, 9(1):128–139, 1947.

[79] G. J. Rattray. *Strategic Warfare in Cyberspace*. The MIT Press, USA, 2001.

[80] A. Rényi. *Foundations of Probability*. Wiley, New York, 1971.

[81] D. J. Robinson. *A course in the theory of groups*. Springer-Verlag New York Inc., 1995.

[82] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. *IEEE Trans. Softw. Eng.*, 22(8):529–551, 1996.

[83] G. Roux. *k-propriétés dans les tableaux de n colonnes; cas particulier de la k-surjectivité et de la k-permutativité*. PhD thesis, University of Paris 6, 1987.

[84] G. Sabidussi. Graphs with given group and given graph-theoretical properties. *Canad. J. Math.*, 9:515–525, 1957.

[85] G. Sabidussi. Graph multiplication. *Math. Z.*, 72:446–457, 1960.

[86] P. Sarkar and P. J. Schellenberg. Construction of symmetric balanced squares with blocksize more than one. *Designs, Codes and Cryptography*, 30(3):235–280, 2003.

[87] G. N. Sárközy. Improved monochromatic loose cycle partitions in hypergraphs. *Discrete Mathematics*, 334:52 – 62, 2014.

[88] P. J. Schellenberg, G. H. J. Van Rees, and S. A. Vanstone. Four pairwise orthogonal latin squares of order 15. *Ars Combinatoria*, 6:141–150, 1978.

[89] J. Seberry and M. Yamada. Hadamard matrices, sequences, and block designs. In *Contemporary Design Theory – A Collection of Surveys (D. J. Stinson and J. Dinitz, Eds.)*, pages 431–560. John Wiley and Sons, 1992.

[90] G. Seroussi and N. H. Bshouty. Vector sets for exhaustive testing of logic circuits. *IEEE Trans. Inf. Theor.*, 34(3):513–522, 1988.

[91] N. Sloane. Covering arrays and intersecting codes. *Journal of Combinatorial Designs*, 1:51–63, 1993.

[92] E. Sperner. Ein Satz über Untermengen einer endlichen Menge. *Math. Z*, 27:544 – 548, 1928.

[93] B. Stevens. *Transversal Covers and Packings*. PhD thesis, University of Toronto, 1998.

[94] B. Stevens, A. Ling, and E. Mendelsohn. A direct construction of transversal covers using group divisible designs. *Ars Combinatoria*, 63:145 – 159, 2002.

[95] B. Stevens, L. Moura, and E. Mendelsohn. Lower bounds for transversal covers. *Designs, Codes and Cryptography*, 15(3):279–299, 1998.

[96] D. R. Stinson. *Combinatorial Designs: Constructions and Analysis*. Springer-Verlag, 2003.

[97] A. P. Street and D. J. Street. *Combinatorics of Experimental Design*. Oxford University Press, Inc., 1986.

[98] D. T. Tang and C. L. Chen. Iterative exhaustive pattern generation for logic testing. *IBM Journal of Research and Development*, 28(2):212–219, 1984.

[99] A. Tong, Y. Wu, and L. Li. Room-temperature phosphorimetry studies of some addictive drugs following dansyl chloride labelling. *Talanta*, 43(9):1429—1436, 1996.

[100] V. G. Vizing. The cartesian product of graphs. *Vyčisl. Sistemy*, 9:30–43, 1963. English translation in Comp. El. Syst., 2:352-365, 1966.

[101] V. I. Voloshin. *Introduction to Graph and Hypergraph Theory*. Nova Science Publishers, Inc., 2009.

[102] M. N. Wegman and J. Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265 – 279, 1981.

[103] B. D. West. *Introduction to graph theory*. Pearson Prentice Hall, New Jersey, 2001.

[104] A. W. Williams. Determination of test configurations for pair-wise interaction coverage. In *13th International Conference on Testing of Communicating Systems (Test-Com), Ottawa, Canada*, pages 59–74, Boston, MA, 2000. Springer US.

[105] C. T. Yu and M. Z. Ozsoyoglu. Algorithm for tree-query membership of a distributed query. In *IEEE Computer Society's International Computer Software and Applications Conference*, pages 306–312. IEEE, 1979.

[106] Z. Zhang, X. Liu, and J. Zhang. Combinatorial testing on ID3v2 Tags of MP3 files. In *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*, pages 587–590, 2012.

Index