Projection - Product and Asymptotic Constructions of Mixed Level Covering Arrays

Gary P. Bazdell

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

School of Mathematics and Statistics
Ottawa-Carleton Institute for Mathematics and Statistics
Carleton University
Ottawa, Ontario, Canada
September 2009

© Copyright 2009

Gary P. Bazdell

Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-60196-9
Our file Notre référence
ISBN: 978-0-494-60196-9

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

■+■
Canada

Abstract

In this thesis, we show that when the parameters k and v of a strength 2 ($t = 2$) covering array are unknown ahead of time, the best single approach for constructing a mixed level covering array is to use a combination of the product construction [26] and projection construction [8] on orthogonal arrays from finite fields. We extend the multiple projection construction for orthogonal arrays to all mixed level covering arrays which satisfy a sub-array and coverage condition. This condition is preserved by the product construction. The product construction and projection construction can very naturally be combined. We show that it is preferable to perform the product construction after the projection construction when the number of disjoint rows are 1 or v and conjecture that this is always the preferred option. In the second part of this thesis we implement our construction with the Combinatorial Test Services library created by IBM Haifa Research Laboratory [14].

Acknowledgements

First and foremost, I would like to thank my supervisor, Brett Stevens, whose encouragement, humor, guidance, impossible to read hand writing, and dedicated support from the initial to the final draft of this thesis enabled me to develop an understanding and appreciation for Design Theory. I would not have continued my education past being an undergraduate student and most certainly would have never made it to the Ph.D. level without his support and guidance. Again, many thanks.

I would also like to thank the other members of the examination committee, Yvan Labiche, Lucia Moura, and Jason Gao for their feedback and suggestions on this thesis

I would also like to thank Colin Weir and Famous Dave Thomson who without their support and drinking games this thesis would never have been completed.

I also wish to thank my mother, Joanne Bazdell, for her love, non-stop support, and patience (also free rent) during the many years of my education.

Finally, I would like to thank the School of Mathematics and Statistics at Carleton University for their financial support and friendly atmosphere. This has been just an amazing time!

Contents

Acknowledgements	i
1 Background - Covering Arrays	1
1.1 Introduction	1
1.2 Organization of Thesis	3
1.3 Miscellaneous Notations and Definitions	4
1.4 Covering Arrays	5
1.5 Generalizations of Covering Arrays	9
1.6 Types of Constructions	12
2 Constructing Covering Arrays	16
2.1 Introduction	16
2.2 General Multiple Projection	17
2.3 Sets of Disjoint Rows	30
2.4 Product Construction for <i>MCA</i> s	30
2.5 Compatibility of the Projection Operation and Product Construction	32
2.5.1 Product after Projection	32
2.5.2 Projection after Product	33
2.6 Project or Product First, That is the Question	34
2.7 Measure of Goodness of CA	39
2.8 Construction Strategies and Comparisons	41
2.8.1 Up to the Next Prime Power	42
2.8.2 Up Two Prime Powers	43

CONTENTS

2.8.3	OAFFs with Disjoint Rows	47
2.8.4	MOLS - Mutually Orthogonal Latin Squares	54
2.9	The Battle of the Century - OAM vs OAFF	55
2.9.1	$v < 10000$	55
2.9.2	Asymptotic Comparison	56
2.10	Summary	60
3	Description of Algorithmic Implementations	61
3.1	Introduction	61
3.2	CTS	62
3.3	Algorithms	63
3.4	Documentation	65
3.5	Summary	70
4	Conclusions and Future Work	72
4.1	Summary of Results	72
4.2	Future Work	74
	Bibliography	76

*Evil will always triumph over
good because good is dumb.*

Dark Helmet, Space Balls

Background - Covering Arrays

1.1 Introduction

Covering arrays have been extensively studied over the last 30 to 40 years [9, 15] and can be found as early as 1948 (Marczewski [22]). They can be found in many applications including: software testing, circuits, networks, testing advanced materials, interactions regulating gene expression, etc [7].

Covering arrays are particularly useful in the design of test suites, which has been the motivation for this thesis, such as the Combinatorial Test Services created by IBM Haifa Research Laboratory [14], TestCover.com [18], TestersDesk.com [19], AETG [5], etc.

The following example illustrates the testing problem and how covering arrays are a mathematical model that can be used to represent a test suite. Let C be a system that is dependent on the input of k different parameters (factors). If we construct an array such that each column represents one of the k parameters and each row is a test of the system and each entry represents

a value for that parameter in that test, then we have a mathematical model of testing the system.

If each of the k parameters could take only two values then the total number of tests one would need to execute to exhaustively verify this system would be 2^k . It is easy to see that even for such a simple system, as k becomes even reasonably large, the number of tests needed would require a great deal of time and resources.

In this model, a covering array is a array which represents a set of tests that guarantee a certain coverage of the system. Formally, a covering array $CA(N; t, k, v)$ is an $N \times k$ array such that in each $N \times t$ sub-array formed by taking t columns of the array, each t -tuple occurs at least once. The value t is the strength of the array for covering interactions, k is the number of parameters, and v is the number of symbols for each factor. Rather than requiring v^k tests, it can be shown that covering arrays require no more than $v^t \log k$ tests [3].

In order to have the model of a covering array reflect a real world testing application, some natural generalizations arise. For example, the system may have k factors but for each factor, say i , the number of possible values it accepts is v_i . Such a system could be represented by a mixed level covering array. Other problems that can occur are: covering arrays with a maximum number of rows (budget constraints), covering array extensions (extending a given set of test vectors), covering array reductions (sub-arrays which retain the coverage property), covering arrays with forbidden configurations, covering arrays on graphs [15, 24].

As covering arrays can represent a test suite, designing a test suite requires

one to have many construction methods for covering arrays in their “toolbox”. The objective of this thesis is to show that when the parameters k and v for $t = 2$ of a covering array are unknown ahead of time, the best single approach for constructing a mixed level covering array is to use a combination of the product construction [26] and projection construction [8] on orthogonal arrays from finite fields. We use the term “unknown ahead of time” to indicate that we want a construction capable of handling any arbitrary k and v at any time. Although it is unrealistic to assume that when constructing a mixed level covering array one would be limited to only a single approach, most other methods have a restricted range of applicability so some universal method is needed and it should be chosen wisely.

1.2 Organization of Thesis

For the remainder of this Chapter, we cover the necessary background on covering arrays needed for this thesis and discuss some of their constructions and generalizations. In Chapter 2, we look at two methods for constructing orthogonal arrays, by sets of mutually orthogonal latin squares and by finite fields. We begin by generalizing the multiple projection construction found in [8] for orthogonal arrays to all mixed level covering arrays that contain a specific sub-array and the needed symbol coverage in a set of rows. We then use this to compare the two methods. In Chapter 3, we use the results from Chapter 2 to develop an algorithm for constructing mixed level covering arrays that can be used by itself or in combination with the Combinatorial Test Service library developed by IBM Haifa Research Laboratory [14]. In

1.3. MISCELLANEOUS NOTATIONS AND DEFINITIONS

Chapter 4, we conclude by outlining the main findings in this thesis and pointing out some future work.

1.3 Miscellaneous Notations and Definitions

This section contains definitions and specific notation that is used throughout this thesis.

Definition 1.1. A *test case* is a k -tuple that has been assigned a single value from each of the k parameters.

Definition 1.2. A *test suite* is a collection of test cases that are intended to be used to test a system with k parameters to verify some specified set of its behaviors.

Definition 1.3. A subsequence of a row in an array is called **Latin** if each symbol in that subsequence occurs exactly once.

Definition 1.4. A subsequence of a column in an array is called **Latin** if each symbol in that subsequence occurs exactly once.

Definition 1.5. The **alphabet** of a parameter is the set of symbols that can occur for that parameter. The size of the alphabet in column i is v_i and will usually be $\{0, 1, \dots, v_i - 1\}$.

Definition 1.6. An *interaction* is a subset of the columns and for each corresponding parameter a choice from its alphabet.

Going back to our model for testing, a test can fail when interactions between parameters are faulty. The goal of the covering array is to contain all possible interactions of a given size.

Notation:

1. $\log$ will mean $\log_2$ unless otherwise stated.
2. $\lceil x \rceil$ will denote the smallest integer $\geq x$.

1.4 Covering Arrays

The definitions in this section come from either [7] or [9].

Definition 1.7. *Let N, t, k , and v be positive integers. An orthogonal array, denoted by $OA(N = v^t; t, k, v)$, is an $N \times k$ array with entries from $\mathbb{Z}_v$ such that in any t distinct columns all t -tuples can be found in exactly one row of the array.*

Looking back at the testing problem, having the requirement that every t -tuple be covered exactly once would be too restrictive when the problem only requires that each t -tuple is covered at least once. Furthermore, OAs do not exist for many parameter sets so this is something we would naturally need to consider. For example, when $t = 2$, $k \leq v + 1$ by the Bose-Bush bound [9].

If we relax the definition of an orthogonal array such that we only require all t -tuples to be found in at least one row of the array, we have the definition of a covering array.

Definition 1.8. *Let N, t, k , and v be positive integers. A covering array, denoted by $CA(N; t, k, v)$, is an $N \times k$ array with entries from $\mathbb{Z}_v$ such that in any t distinct columns all t -tuples can be found in at least one row of the array.*

1.4. COVERING ARRAYS

Example 1.9. *The following is an example of a $CA(9; 2, 4, 3)$*

$$\begin{array}{cccc}
 0 & 0 & 0 & 0 \\
 0 & 1 & 1 & 2 \\
 1 & 1 & 0 & 1 \\
 1 & 2 & 1 & 0 \\
 2 & 1 & 2 & 0 \\
 2 & 2 & 0 & 2 \\
 0 & 2 & 2 & 1 \\
 2 & 0 & 1 & 1 \\
 1 & 0 & 2 & 2
 \end{array}$$

Definition 1.10. *The covering array is **optimal** if it contains the minimum possible number of rows. The size of an optimal covering array is the **covering array number** $CAN(t, k, v)$.*

From the definition of an orthogonal array, each t -tuple can be found exactly once in a row of the array. Thus an orthogonal array is an optimal covering array.

Proposition 1.11. *Suppose there exists an $OA(N; t, k, v)$. Then the covering array number $CAN(t, k, v) = N = v^t$.*

Using basic counting arguments we obtain the following.

Proposition 1.12. *For all $CA(N; t, k, v)$*

$$v^t \leq CAN(t, k, v) \leq v^k$$

Example 1.13. *The $CA(9; 2, 4, 3)$ in Example 1.9 has $v^t = 3^2 = 9$ rows, hence is optimal.*

1.4. COVERING ARRAYS

Exact optimal covering array sizes have been found in general only for the case $t = 2 = v$. This result has been independently determined several times (Chandra, Kou, Markowsky, and Zaks [4], Katona [20], Kleitman and Spencer [21], and Rényi [28]).

Theorem 1.14. [4, 20, 21, 28] *For all $k > 1$, when $t = v = 2$, N grows as follows. If*

$$f(N) = \binom{N-1}{\lceil \frac{N}{2} \rceil - 1}$$

then

$$CAN(2, k, 2) = \min\{N : f(N) \geq k\}$$

For covering arrays with $t = 2$, Gargano, Körner, and Vaccaro [12] gave the following asymptotic result for $v \geq 2$.

Theorem 1.15. [12] *Let $v \geq 2$ be a positive integer. Then*

$$\lim_{k \rightarrow \infty} \frac{CAN(2, k, v)}{\log k} = \frac{v}{2}$$

Example 1.16. (Testing Problem) *Suppose Microsoft developed a new software package and before making it available to the public they wanted to be sure that it would work properly with certain types of monitors, operating systems, memory, and video cards so that it wouldn't be as big of an embarrassment as Windows Vista. Table 1.1 lists the parameters and the possible values they can take.*

1.4. COVERING ARRAYS

Parameter 1	Parameter 2	Parameter 3	Parameter 4
Monitor	Operating System	Memory	Video Card
Acer (0)	Windows (0)	Corsair (0)	Asus (0)
LG (1)	Mac (1)	Kingston (1)	Sapphire (1)
Samsung (2)	Linux (2)	OCZ (2)	XFX (2)

Table 1.1: System Parameters

To test this system exhaustively, one would need to perform $v^k = 3^4 = 81$ tests. This could be possible, but if we extend this system to 10 parameters with 5 possible values each, one would need to perform $5^{10} = 9765625$ tests.

Cohen, Dalal, Fredman and Patton [5] gave strong evidence that the majority of defects are due to the interaction between two parameters. If we focused on interactions that involve pairs of parameters, the number of tests would go from 81 to $3^2 = 9$ tests. Using Example 1.9, which is an optimal covering array, as our mathematical model for this example, if we relabel the parameters with the parameters given in Table 1.1, then Table 1.2 can cover all interactions that involve pairs of parameters for this example.

1.5. GENERALIZATIONS OF COVERING ARRAYS

Test Case	Monitor	Operating System	Memory	Video Card
1	Acer	Windows	Corsair	Asus
2	Acer	Mac	Kingston	XFX
3	LG	Mac	Corsair	Sapphire
4	LG	Linux	Kingston	Asus
5	Samsung	Mac	OCZ	Asus
6	Samsung	Linux	Corsair	XFX
7	Acer	Linux	OCZ	Sapphire
8	Samsung	Windows	Kingston	Sapphire
9	LG	Windows	OCZ	XFX

Table 1.2: Test Suite for Software Example

1.5 Generalizations of Covering Arrays

One of the main generalizations of a covering array is the mixed level covering array (*MCA*). The *MCA* is used when we require that the alphabet sizes may vary.

Definition 1.17. A mixed level covering array $MCA(N; t, k, (v_1, \dots, v_k))$ is an $N \times k$ array where column i is filled from alphabet v_i . Let $\{i_1, \dots, i_t\} \subset \{1, \dots, k\}$, and consider the subarray of size $N \times t$ obtained by selecting columns $i_1, \dots, i_t$ of the *MCA*. There are $\prod_{j=1}^t v_{i_j}$ distinct t -tuples that could appear as a row, and an *MCA* requires that each appears at least once.

Example 1.18. The following is an example of a $MCA(9; 2, 4, (3, 3, 2, 2))$

1.5. GENERALIZATIONS OF COVERING ARRAYS

$$\begin{array}{cccc}
 0 & 0 & 0 & 0 \\
 0 & 1 & 1 & 1 \\
 1 & 2 & 0 & 1 \\
 2 & 2 & 0 & 0 \\
 1 & 1 & 1 & 0 \\
 0 & 2 & 1 & 1 \\
 1 & 0 & 0 & 0 \\
 2 & 0 & 1 & 1 \\
 2 & 1 & 0 & 1
 \end{array}$$

Definition 1.19. *The covering array is **optimal** if it contains the minimum possible number of rows. The size of an optimal mixed level covering array is the **covering array number** $CAN(t, k, (v_1, \dots, v_k))$.*

Using basic counting arguments we obtain the following.

Proposition 1.20. *For all $MCA(N; t, k, (v_1, \dots, v_k))$ and $v_1 \leq v_2 \leq \dots \leq v_k$*

$$v_{k-t+1} \times \dots \times v_{k-1} \times v_k \leq CAN(t, k, (v_1, \dots, v_k)) \leq v_1 \times \dots \times v_k$$

Example 1.21. *Example 1.18 has covering array number 9 since the two largest alphabet sizes are both 3 and $3^2 = 9 = N$.*

Other natural generalizations of the Testing problem can be found in [11, 15, 23, 24]. The following are a few generalization examples:

1. Covering arrays with a maximum number of rows

In smaller testing environments the resources (time, cost, etc) could

1.5. GENERALIZATIONS OF COVERING ARRAYS

be strictly limited. One could model this situation by considering the problem of the best possible test suite within a fixed number of test cases.

2. **Covering array extension**

Suppose a test suite was constructed but new factors were needed to be added (i.e. a new software upgrade). To continue using the old test suite, one would look at the minimal extension needed to satisfy the coverage property on the new factors with the old factors.

3. **Covering array reduction**

Suppose a test suite was constructed but factors were needed to be removed. One would look at the minimal number of tests needed from the test suite to cover the new subset of factors.

4. **Covering arrays with forbidden configurations**

Suppose there was a system in which some input combinations are invalid. One would need to design a test suite where certain pairwise interactions would never appear in a test while all others must be covered.

5. **Error Locating Arrays**

For each test that we run there are two possible outcomes, pass or fail. Problems arise when a test returns a fail. The reasons for this is we cannot be certain which interaction was the cause of the failure. An Error Locating Array is a covering array that has enough tests with the right overlap so that we can do some kind of analysis to find all the faulty pairs.

6. Covering arrays on graphs

Suppose there was a factor that did not require to be covered by all other factors. Covering arrays on graphs only require that specified pairs of factors are covered and these pairs can be recorded in a graph structure.

1.6 Types of Constructions

In this section, we give a brief introduction to three of the main methods for constructing covering arrays. A more complete survey of construction methods can be found in [7] and [13].

There are three main methods for constructing covering arrays:

1. Direct constructions
2. Recursive constructions
3. Metaheuristics

Each construction method has its own benefits and weaknesses for specific parameters N, t, k and v .

Direct constructions use algebraic and combinatorial methods such as orthogonal arrays [16] as we have seen previously. Orthogonal arrays produce optimal covering arrays but are limited by bose-bush bound $k \leq v + 1$ [9]. Algebraic group methods require the existence of difference covering matrices which are only known to exist for restricted range of parameters [25, 33]. Recursive constructions use the existence of smaller covering arrays to build larger ones. These include Wilson-type theorems [32] which keeps k constant

1.6. TYPES OF CONSTRUCTIONS

and increase v , Roux-type [29] which is a type of product construction that keeps v fixed and increase k , Ordered Designs [3] which require the existence of a $CA(3, q + 1, 2)$ and a $OD(3, q + 1, q + 1)$ to construct a $CA(3, q + 1, q + 1)$ where q is a prime power, and many more [7, 31, 32]. Since direct and recursive constructions have restricted range of parameters or require small, good ingredients, a lot of attention has been focused on metaheuristic techniques.

Metaheuristics are typically used when there is no known method to find an optimal solution, under the given constraints or at all. The following are a few example of popular metaheuristics.

Greedy Algorithms and Variations is any algorithm that follows the problem solving metaheuristic of making the locally optimal choice at each stage with the hope of finding the globally good solution [3, 7, 30].

Example 1.22. *Start with an empty covering array and build one row at a time and continue until all pairs have been covered. The Greedy Algorithm concentrates on building each new row such that it contains more previously uncovered pairs.*

A good example of this can be found in [3].

Hill Climbing can be used to solve problems that have many solutions, some of which are better than others. It starts with a random (potentially poor) solution, and iteratively makes small changes to the solution, each time improving it a little [7, 30]

Example 1.23. *Start with an array in which not all factors are covered. Select one row and replace a random symbol from one of the*

1.6. TYPES OF CONSTRUCTIONS

factors by another possible symbol for that factor. If at this stage the new array is an improvement on the last, keep new array and continue.

Tabu Search enhances the performance of a local search method by using memory structures: once a potential change has been determined, it is marked as “tabu” so that the algorithm does not visit that possibility repeatedly [7, 27]

Example 1.24. *Start with an array in which not all factors are covered. Select one uncovered t -combination at random. Check which rows require only the change of a single element such that it would cover the t -combination. Select the best of all the found rows and replace with new symbol. This potential change is marked as “tabu” and recorded into the “tabu” list as to avoid loops.*

Simulated Annealing uses the same approach as hill climbing but allows the algorithm to make choices that reduce the quality of the current solution. [7, 30, 31]

Example 1.25. *Using the above example, if the selected change is not as good as the current array the heuristic guarantees that at each step, there is a non-zero probability of moving to a state that is worse than the current.*

Genetic Algorithm maintain a population of putative solutions, and to evolve the population from one generation to the next. [7, 30]

Example 1.26. *The Genetic Algorithm starts with a “population” of potential arrays. These arrays are grouped into pairs of parent “genes”*

1.6. TYPES OF CONSTRUCTIONS

and combined with one another. The same move used in the other two algorithms is applied to each of the new arrays to simulate mutation. The better arrays are kept and the others lost and the process continues over again.

In general the complexity constraint restricts them to small parameters.

Two fundamental goals in designing testing suites are finding covering array constructions with provably good run times and with provably good or optimal covering array numbers. Metaheuristic techniques abandon one or both of these goals.

As stated in the introduction to this thesis, it is unrealistic to assume that when constructing a mixed level covering array one would be limited to only a single approach. Since there are few direct constructions, recursive constructions rely on the existence of smaller constructions, and metaheuristics techniques may construct something very far from optimal, a fundamental tool in any testing suite “toolbox” would be a construction which can work for any set of parameters.

In the following Chapter we define the tools needed for our single approach construction.

*I'm not bad, I'm just drawn
that way.*

Who Framed Roger Rabbit

2

Constructing Covering Arrays

2.1 Introduction

The focus of this chapter will be on two strategies for constructing *MCA*s:

1. *OAs* from *MOLS* (*OAM*)
2. *OAs* from finite fields (*OAFF*)

combined with the product and projection constructions. This chapter will argue that if the parameters k and v for $t = 2$ are unknown ahead of time (arbitrary k and v), the best single approach for constructing *MCA*s is to use a combination of the product construction and the projection construction on *OAFF*s.

The organization of this chapter is as follows: in Section 2, we begin by generalizing the multiple projection construction for *OAs* found in [8] to any *MCA* that contains a specified sub-array and the needed symbol coverage in a set of rows. In Sections 3 and 4, we define what a set of disjoint rows are

2.2. GENERAL MULTIPLE PROJECTION

and state the product construction for an *MCA*. If we apply the product construction to an *MCA* which has a set of disjoint rows, we show that, with a simple relabeling of the alphabets, we can remove rows and still have an *MCA*. In Section 5, we state the requirements needed to combine the product and projection constructions. In Section 6, we determine that it is always best to project before using the product construction with *MCAs* that have sets of disjoint rows of size 1 and $v_1 = \min\{v_1, \dots, v_k\}$.

In Section 7, we derive a measure of the “goodness” of a *MCA* from the parameters k and N that can be used to compare different *CAs*. In Section 8, we compare different strategies for constructing *OAFFs* combined with different combinations of product constructions. We define the construction of *OAMs*. Finally, in Section 9, we compare directly the projection-product construction on *OAFFs* and the product construction on *OAMs* for values of $v < 10000$ and prove that for all v , with the best current known asymptotic growth rate for the number of *MOLS*, the projection-product *OAFF* constructions will always yield a smaller covering array. For values of v that are prime powers, the two methods of construction are equivalent.

2.2 General Multiple Projection

In this section we will generalize the multiple projection construction used in [8] on *OAs* to any *MCA* satisfying specific conditions. The general idea of this construction is that we can take an *MCA* and remove the same number of symbols from each v_i , while at the same time increasing the size of k by that same amount. The utility of this property will become clear as we

2.2. GENERAL MULTIPLE PROJECTION

combine construction methods later in this chapter.

We will first need a few definitions.

Definition 2.1. [8] Let i and j be two columns, $i \neq j$ in an $MCA(N; 2, k, (v_1, \dots, v_k))$ and σ_i and σ_j be symbols in column i and j in the same row. Then $\{(i, \sigma_i), (j, \sigma_j)\}$ is a **private pair** if the ordered pair either has σ_i or σ_j equal to $\star$ where $\star$ denotes a “do not care” symbol that can be used as parameter value, or occurs exactly once in the $N \times 2$ sub-array formed by columns i and j .

Definition 2.2. [8] Let $I = \{i_1, \dots, i_s\}$ be a set of column indices. A row of an $MCA(N; 2, k, (v_1, \dots, v_k))$ is an **I -private** row if every pair of symbols in the row indexed by any $\{i, j\} \subseteq I$, $i \neq j$, is private

In [8], Colbourn defines a projection operation on an MCA :

Theorem 2.3. [8] Let C be an $MCA(N; 2, k, (v_1, \dots, v_k))$. Let I be a set of $s \leq k$ column indices. Suppose that the p th row is an I -private row. Suppose further that the symbol x_i in row p and column $i \in I$ is not $\star$, and appears in at least v_i rows of C . Then there is an $MCA(N-1; 2, k+1, (v_1-1, \dots, v_k-1, s))$.

He then goes on to prove that the projection operation can be applied multiple times to an orthogonal array constructed from a finite field:

Theorem 2.4. [8] Let q be a prime power. Let $1 \leq m \leq q$ and $1 \leq s \leq q-m$. Then there is an $MCA(q^2 - m; 2, q + 1 + m, (q - m)^{q+1}, s^m)$. In particular, there exists a $CA(q^2 - m; 2, q + 1 + m, q - m)$.

The following definition states the conditions needed in order to use the projection construction multiple times on a general MCA .

2.2. GENERAL MULTIPLE PROJECTION

Definition 2.5. Let C be an $MCA(N; 2, k, (v_1, \dots, v_k))$ such that $v_1 \leq v_2 \leq \dots \leq v_k$. Let $I = \{i_1, \dots, i_s\}$ be a set of column indices, $s \leq k$. Let C_{proj} be a $m \times s$ sub-array formed by $1 \leq m \leq v_1$ rows and $1 \leq s \leq k$ columns of C . If each row of C_{proj} is I -private and each column of C_{proj} is Latin then C_{proj} is said to be a $m \times s$ **projective sub-array**.

Theorem 2.6. Let C be an $MCA(N; 2, k, (v_1, \dots, v_k))$ which contains a $m \times s$ projective sub-array. Let $1 \leq m' \leq m$ and $1 \leq s' \leq s - m'$. Suppose that the symbol x_i in each row and column $1 \leq i \leq s'$ of the sub-array is not $\star$ and appears in at least $v_i - m'$ other rows of C . Then there is an $MCA(N - m'; 2, k + m', (v_1 - m', \dots, v_k - m'), (s')^{m'})$.

Proof. Refer to Algorithm 2.1 for construction method.

Correctness of the algorithm: Because the rows in the projective sub-array are I -private, when we partition C into classes for some column, those classes are guaranteed to be disjoint. The columns in the projective sub-array are Latin so when we partition C , the rows not being used in the sub-array will be placed in D and left unchanged in the s' columns of I . By hypothesis each symbol x_i , $i \in I$, in row r_k , $k \in R$ appears in at least $v_i - m'$ rows of C in addition to C_{proj} so each symbol i , added in will be matched with a permutation of the remaining symbols in column i (A).

We now check that for any two columns i and j , all pairs of symbols are covered. If $1 \leq i < j \leq k$, columns i and j have gone unchanged from the original MCA except for the removal or replacement of the deleted symbols. So all remaining symbol pairs are covered. We now have two cases to consider.

Case 1: Suppose j is one of the new columns added and $i \leq k$. Let σ_j

2.2. GENERAL MULTIPLE PROJECTION

Algorithm 2.1 General Multiple Projection

Let $I = \{l_1, \dots, l_{s'}\}$ be column indices from the projective sub-array

Let $R = \{r_1, \dots, r_{m'}\}$ be row indices from the projective sub-array

for each row r_j in R **do**

 Let $(x_1, \dots, x_k)$ be the row r_j in C

 Let $C_1, \dots, C_{s'}$ and D be partition classes, initially empty.

for $i = 1$ to s' **do**

 Place all rows in C , except r_j , that contain x_{l_i} in column l_i into C_i

end for

 Place all remaining rows in D

for $i = 1$ to s' **do**

for each row $r \in C_i$ **do**

 Place symbol i in the new column $k + j$

end for

for each C_i **do**

 Replace the l_i th column with a permutation of the remaining symbols
 in that column other than the m' in R and all other entries $\star$. (A)

end for

end for

for each column g **do**

 Replace all symbols equal to x_g in column g to $\star$

end for

 In all rows of D , place $\star$ in new $k + 1$ column. Delete row r_j .

end for

for each row in the sub-array not in R , $i = 1$ **do**

 Replace all $\star$ symbols in the added columns to i and set $i = i + 1$ (B)

end for

2.2. GENERAL MULTIPLE PROJECTION

be a specific symbol in column j and σ_i be a specific symbol in column i . If we look at the partition C_{σ_j} we have two possibilities. One is that i was the l_{σ_j} column. Then by hypothesis, all symbols x_i of column l_{σ_j} were replaced with a permutation of the remaining symbols in that column. Since σ_i is a remaining symbol, there exists a row in C such that $\{(i, \sigma_i), (j, \sigma_j)\}$ exists. If i was not the l_{σ_j} column, because C is an *MCA* there was a row that contains $\{(i, \sigma_i)(l_{\sigma_j}, x_{l_{\sigma_j}})\}$ so that same row now contains $\{(i, \sigma_i)(j, \sigma_j)\}$.

Case 2: Suppose i and j are both new columns added. Let σ_j be a specific symbol in column j and σ_i be a specific symbol in column i . If $\sigma_j = \sigma_i$ then we have changed the remaining rows in the projective sub-array which were extended by $\star$ positions (B) to ensure the presence of the pair. If $\sigma_j \neq \sigma_i$ because C is a *MCA* we know that there is a row that contains the pair $\{(l_{\sigma_j}, x_{l_{\sigma_j}}), (l_{\sigma_i}, x_{l_{\sigma_i}})\}$ and thus this same row will now contain $\{(j, \sigma_j), (i, \sigma_i)\}$. $\square$

Example 2.7. *The following example illustrates multiple projection on a $CA(19; 2, 6, 4)$, say C . Choosing rows 1, 2, 8, 14 and columns 4 and 6 gives us a 4×2 projective sub-array, C_{proj} .*

2.2. GENERAL MULTIPLE PROJECTION

$$\begin{array}{cccccc}
 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & 1 & 2 & 2 & 3 & 2 \\
 2 & 0 & 1 & 2 & 2 & 3 \\
 3 & 2 & 0 & 1 & 2 & 2 \\
 2 & 3 & 2 & 0 & 1 & 2 \\
 2 & 2 & 3 & 2 & 0 & 1 \\
 1 & 2 & 2 & 3 & 2 & 0 \\
 0 & 2 & 3 & 3 & 1 & 3 \\
 3 & 0 & 2 & 3 & 3 & 1 \\
 C = & 1 & 3 & 0 & 2 & 3 & 3 & C_{proj} = & 0 & 0 \\
 & 3 & 1 & 3 & 0 & 2 & 3 & & 2 & 2 \\
 & 3 & 3 & 1 & 3 & 0 & 2 & & 3 & 3 \\
 & 2 & 3 & 3 & 1 & 3 & 0 & & 1 & 1 \\
 & 0 & 3 & 1 & 1 & 2 & 1 & & & \\
 & 1 & 0 & 3 & 1 & 1 & 2 & & & \\
 & 2 & 1 & 0 & 3 & 1 & 1 & & & \\
 & 1 & 2 & 1 & 0 & 3 & 1 & & & \\
 & 1 & 1 & 2 & 1 & 0 & 3 & & & \\
 & 3 & 1 & 1 & 2 & 1 & 0 & & &
 \end{array}$$

We need to choose an m' and s' such that $m' \leq m = 4$ and $s' \leq m - m' = 4 - m'$. Let $m' = 2$, $s' = 2$ so we can choose $I = \{4, 6\}$ and $R = \{1, 2\}$. Let $r_1 = (0, 0, 0, 0, 0, 0)$. For all rows in C except r_1 , partition C into 3 classes, C_1, C_2 , and D . C_1 contains all the rows that have a 0 in column 4, C_2 contains all the rows that have a 0 in column 6, and D has the remaining rows.

2.2. GENERAL MULTIPLE PROJECTION

$$\begin{array}{rcccl}
 & & & & 0 & 1 & 2 & 2 & 3 & 2 \\
 & & & & 2 & 0 & 1 & 2 & 2 & 3 \\
 & & & & 3 & 2 & 0 & 1 & 2 & 2 \\
 & & & & 2 & 2 & 3 & 2 & 0 & 1 \\
 & & & & 0 & 2 & 3 & 3 & 1 & 3 \\
 C_1 = \begin{array}{cccccc} 2 & 3 & 2 & 0 & 1 & 2 \\ 3 & 1 & 3 & 0 & 2 & 3 \\ 1 & 2 & 1 & 0 & 3 & 1 \end{array} & C_2 = \begin{array}{cccccc} 1 & 2 & 2 & 3 & 2 & 0 \\ 2 & 3 & 3 & 1 & 3 & 0 \\ 3 & 1 & 1 & 2 & 1 & 0 \end{array} & D = \begin{array}{cccccc} 3 & 0 & 2 & 3 & 3 & 1 \\ 1 & 3 & 0 & 2 & 3 & 3 \\ 3 & 3 & 1 & 3 & 0 & 2 \\ 0 & 3 & 1 & 1 & 2 & 1 \\ 1 & 0 & 3 & 1 & 1 & 2 \\ 2 & 1 & 0 & 3 & 1 & 1 \\ 1 & 1 & 2 & 1 & 0 & 3 \end{array}
 \end{array}$$

Add a column 7 to each C_1, C_2 , and D . In C_1 , replace the entries in the 4th column with a permutation of the symbols for that column that are not 0 or 2. Then, in the 7th column, add 1 in each entry. In C_2 , replace the entries in the 6th column with a permutation of the symbols for that column that are not 0 or 2. Then, in the 7th column, add 2 in each entry. In the 7th column of D , add $\star$ in each entry. For C_1, C_2 , and D , for each entry in column i , replace any symbol that is equal to the symbol in column i in r_1 with $\star$. This gives us:

$$\begin{array}{rcccl}
 & & & & 2 & 3 & 2 & 1 & 1 & 2 & 1 & & 1 & 2 & 2 & 3 & 2 & 1 & 2 \\
 C_1 = \begin{array}{cccccc} 2 & 3 & 2 & 1 & 1 & 2 & 1 \\ 3 & 1 & 3 & 3 & 2 & 3 & 1 \\ 1 & 2 & 1 & \star & 3 & 1 & 1 \end{array} & C_2 = \begin{array}{cccccc} 1 & 2 & 2 & 3 & 2 & 1 & 2 \\ 2 & 3 & 3 & 1 & 3 & 3 & 2 \\ 3 & 1 & 1 & 2 & 1 & \star & 2 \end{array}
 \end{array}$$

2.2. GENERAL MULTIPLE PROJECTION

$$\begin{array}{cccccccc}
 & \star & 1 & 2 & 2 & 3 & 2 & \star \\
 & 2 & \star & 1 & 2 & 2 & 3 & \star \\
 & 3 & 2 & \star & 1 & 2 & 2 & \star \\
 & 2 & 2 & 3 & 2 & \star & 1 & \star \\
 & \star & 2 & 3 & 3 & 1 & 3 & \star \\
 D = & 3 & \star & 2 & 3 & 3 & 1 & \star \\
 & 1 & 3 & \star & 2 & 3 & 3 & \star \\
 & 3 & 3 & 1 & 3 & \star & 2 & \star \\
 & \star & 3 & 1 & 1 & 2 & 1 & \star \\
 & 1 & \star & 3 & 1 & 1 & 2 & \star \\
 & 2 & 1 & \star & 3 & 1 & 1 & \star \\
 & 1 & 1 & 2 & 1 & \star & 3 & \star
 \end{array}$$

We now delete r_1 . Putting the partitions back together gives us a new C , say C'

2.2. GENERAL MULTIPLE PROJECTION

$$C' = \begin{array}{cccccc} \star & 1 & 2 & 2 & 3 & 2 & \star \\ 2 & \star & 1 & 2 & 2 & 3 & \star \\ 3 & 2 & \star & 1 & 2 & 2 & \star \\ 2 & 3 & 2 & 1 & 1 & 2 & 1 \\ 2 & 2 & 3 & 2 & \star & 1 & \star \\ 1 & 2 & 2 & 3 & 2 & 1 & 2 \\ \star & 2 & 3 & 3 & 1 & 3 & \star \\ 3 & \star & 2 & 3 & 3 & 1 & \star \\ 1 & 3 & \star & 2 & 3 & 3 & \star \\ 3 & 1 & 3 & 3 & 2 & 3 & 1 \\ 3 & 3 & 1 & 3 & \star & 2 & \star \\ 2 & 3 & 3 & 1 & 3 & 3 & 2 \\ \star & 3 & 1 & 1 & 2 & 1 & \star \\ 1 & \star & 3 & 1 & 1 & 2 & \star \\ 2 & 1 & \star & 3 & 1 & 1 & \star \\ 1 & 2 & 1 & \star & 3 & 1 & 1 \\ 1 & 1 & 2 & 1 & \star & 3 & \star \\ 3 & 1 & 1 & 2 & 1 & \star & 2 \end{array}$$

Let $r_2 = (\star, 1, 2, 2, 3, 2, \star)$. For all rows in C' except r_2 , partition C' into 3 classes, C_1 , C_2 , and D . C_1 contains all the rows that have a 2 in column 4, C_2 contains all the rows that have a 2 in column 6, and D has the remaining rows.

2.2. GENERAL MULTIPLE PROJECTION

$$C_1 = \begin{array}{cccccc} 2 & \star & 1 & 2 & 2 & 3 & \star \\ 2 & 2 & 3 & 2 & \star & 1 & \star \\ 1 & 3 & \star & 2 & 3 & 3 & \star \\ 3 & 1 & 1 & 2 & 1 & \star & 2 \end{array} \quad C_2 = \begin{array}{cccccc} 3 & 2 & \star & 1 & 2 & 2 & \star \\ 2 & 3 & 2 & 1 & 1 & 2 & 1 \\ 3 & 3 & 1 & 3 & \star & 2 & \star \\ 1 & \star & 3 & 1 & 1 & 2 & \star \end{array}$$

$$D = \begin{array}{cccccc} 1 & 2 & 2 & 3 & 2 & 1 & 2 \\ \star & 2 & 3 & 3 & 1 & 3 & \star \\ 3 & \star & 2 & 3 & 3 & 1 & \star \\ 3 & 1 & 3 & 3 & 2 & 3 & 1 \\ 2 & 3 & 3 & 1 & 3 & 3 & 2 \\ \star & 3 & 1 & 1 & 2 & 1 & \star \\ 2 & 1 & \star & 3 & 1 & 1 & \star \\ 1 & 2 & 1 & \star & 3 & 1 & 1 \\ 1 & 1 & 2 & 1 & \star & 3 & \star \end{array}$$

Add a column 8 to each C_1, C_2 , and D . In C_1 , replace the entries in the 4th column with a permutation of the symbols for that column that are not 0 or 2. Then, in the 8th column, add 1 in each entry. In C_2 , replace the entries in the 6th column with a permutation of the symbols for that column that are not 0 or 2. Then, in the 8th column, add 2 in each entry. In the 8th column of D , add $\star$ in each entry. For C_1, C_2 , and D , for each entry in column i , replace any symbol that is equal to the symbol in column i in r_2 with $\star$. This gives us:

2.2. GENERAL MULTIPLE PROJECTION

$$\begin{array}{rcl}
 & 2 & \star & 1 & 1 & 2 & 3 & \star & 1 & & 3 & 2 & \star & 1 & 2 & 1 & \star & 2 \\
 C_1 = & 2 & 2 & 3 & 3 & \star & 1 & \star & 1 & & C_2 = & 2 & 3 & \star & 1 & 1 & 3 & 1 & 2 \\
 & 1 & 3 & \star & \star & \star & 3 & \star & 1 & & & 3 & 3 & 1 & 3 & \star & \star & \star & 2 \\
 & 3 & \star & 1 & \star & 1 & \star & 2 & 1 & & & 1 & \star & 3 & 1 & 1 & \star & \star & 2
 \end{array}$$

$$\begin{array}{rcl}
 & 1 & 2 & \star & 3 & 2 & 1 & 2 & \star \\
 & \star & 2 & 3 & 3 & 1 & 3 & \star & \star \\
 & 3 & \star & \star & 3 & \star & 1 & \star & \star \\
 & 3 & \star & 3 & 3 & 2 & 3 & 1 & \star \\
 D = & 2 & 3 & 3 & 1 & \star & 3 & 2 & \star \\
 & \star & 3 & 1 & 1 & 2 & 1 & \star & \star \\
 & 2 & \star & \star & 3 & 1 & 1 & \star & \star \\
 & 1 & 2 & 1 & \star & \star & 1 & 1 & \star \\
 & 1 & \star & \star & 1 & \star & 3 & \star & \star
 \end{array}$$

We now delete r_2 . Putting the partitions back together gives us a new C' , say C''

2.2. GENERAL MULTIPLE PROJECTION

$$\begin{array}{cccccccc}
 2 & \star & 1 & 1 & 2 & 3 & \star & 1 \\
 3 & 2 & \star & 1 & 2 & 1 & \star & 2 \\
 2 & 3 & \star & 1 & 1 & 3 & 1 & 2 \\
 2 & 2 & 3 & 3 & \star & 1 & \star & 1 \\
 1 & 2 & \star & 3 & 2 & 1 & 2 & \star \\
 \star & 2 & 3 & 3 & 1 & 3 & \star & \star \\
 3 & \star & 1 & \star & 1 & \star & 2 & 1 \\
 1 & 3 & \star & \star & \star & 3 & \star & 1 \\
 C'' = & 3 & \star & 3 & 3 & 2 & 3 & 1 & \star \\
 & 3 & 3 & 1 & 3 & \star & \star & \star & 2 \\
 & 2 & 3 & 3 & 1 & \star & 3 & 2 & \star \\
 & \star & 3 & 1 & 1 & 2 & 1 & \star & \star \\
 & 1 & \star & 3 & 1 & 1 & \star & \star & 2 \\
 & 2 & \star & \star & 3 & 1 & 1 & \star & \star \\
 & 1 & 2 & 1 & \star & \star & 1 & 1 & \star \\
 & 1 & \star & \star & 1 & \star & 3 & \star & \star \\
 & 3 & \star & \star & 3 & \star & 1 & \star & \star
 \end{array}$$

Notice row 8 and 14 from C which are now rows 6 and 12 in C'' . Since these two rows were in the projective sub-array, their entries in column 7 and 8 are $\star$. In order to have the last two columns cover all 2-tuples, we replace the $\star$ entries in column 7 and 8 of row 6 with 1's and row 12 with 2's.

2.2. GENERAL MULTIPLE PROJECTION

$$\begin{array}{cccccccc}
 2 & \star & 1 & 1 & 2 & 3 & \star & 1 \\
 3 & 2 & \star & 1 & 2 & 1 & \star & 2 \\
 2 & 3 & \star & 1 & 1 & 3 & 1 & 2 \\
 2 & 2 & 3 & 3 & \star & 1 & \star & 1 \\
 1 & 2 & \star & 3 & 2 & 1 & 2 & \star \\
 \star & 2 & 3 & 3 & 1 & 3 & 1 & 1 \\
 3 & \star & 1 & \star & 1 & \star & 2 & 1 \\
 1 & 3 & \star & \star & \star & 3 & \star & 1 \\
 C'' = & 3 & \star & 3 & 3 & 2 & 3 & 1 & \star \\
 & 3 & 3 & 1 & 3 & \star & \star & \star & 2 \\
 & 2 & 3 & 3 & 1 & \star & 3 & 2 & \star \\
 & \star & 3 & 1 & 1 & 2 & 1 & 2 & 2 \\
 & 1 & \star & 3 & 1 & 1 & \star & \star & 2 \\
 & 2 & \star & \star & 3 & 1 & 1 & \star & \star \\
 & 1 & 2 & 1 & \star & \star & 1 & 1 & \star \\
 & 1 & \star & \star & 1 & \star & 3 & \star & \star \\
 & 3 & \star & \star & 3 & \star & 1 & \star & \star
 \end{array}$$

C'' is now a $CA(17; 2, 8, 2)$.

Observation 2.8. *When we chose s columns for the projective sub-array, we did not verify that each entry in column i appears in at least $v_i - m'$ other rows of C . Thus some columns in the projective sub-array may not hold this property and can't be used in the multiple projection construction so we choose s' that can be used. We also require that s' rows are not used in the projective sub-array in order to maintain the coverage property in the*

2.3. SETS OF DISJOINT ROWS

new columns as the constant terms will not be added until step (B) of the algorithm which require these rows.

2.3 Sets of Disjoint Rows

Definition 2.9. Let C be an $MCA(N; 2, k, (v_1, \dots, v_k))$. C is said to have a set of n disjoint rows if there exists n rows in C such that for each i , $1 \leq i \leq k$, column i is Latin in those rows. We then denote such a C by $MCA(N; 2, k, (v_1, \dots, v_k) : n)$

Clearly, for every MCA there exists at least a set of a single disjoint row, which can be any row. The usefulness of disjoint rows will become clear in the following sections when we focus on optimizing constructions. If we omit the value of n when discussing an MCA , we will simply mean $n = 1$.

2.4 Product Construction for MCAs

In this section we define the product construction for $MCAs$.

Theorem 2.10. [26] Let $v_1 \leq v_2 \leq \dots \leq v_k$. If there exists a $MCA(N; 2, k, (v_1, \dots, v_k) : n)$ and $MCA(M; 2, l, v_k : m)$, then there exists a $MCA(N + M - m; 2, kl, (v_1, \dots, v_k)^l : n)$

Proof. Concatenate l copies of the $MCA(N; 2, k, (v_1, \dots, v_k) : n)$, say A , side by side. Without loss of generality, assume the set of m disjoint rows in $MCA(M; 2, l, v_k : m)$ are

2.4. PRODUCT CONSTRUCTION FOR *MCAS*

$$\begin{array}{ccc}
0 & \dots & 0 \\
1 & \dots & 1 \\
\vdots & \vdots & \vdots \\
m-1 & \dots & m-1
\end{array}$$

Delete these rows to get an $MCA(M-m; 2, l, v_k)$, B , such that B now covers all **distinct** pairs in each pair of columns. For the g th copy of A , $1 \leq g \leq l$, take the g -th column of B , and vertically concatenate this column with all columns of the g -th copy of A . For each of the new kl columns, replace the symbols that were not in the original column of A with $\star$.

We now verify that all pairs are still covered. Let i and j be any two columns in the $MCA(N+M; 2, kl, (v_1, \dots, v_k)^l : n)$. If i and j were constructed from same column of A , then all pairs (i, i) are covered in column pairs of A . All other pairs are covered in the 2 concatenated columns of B . If i and j were not constructed from the same column of A , then any pair of symbols in i and j are covered because they were already covered in A . $\square$

Corollary 2.11. *If there exists an $MCA(N; 2, k, v : m)$, then there exists an $MCA(2N-m; 2, k^2, v : m)$.*

The figure below illustrates the product construction.

2.5. COMPATIBILITY OF THE PROJECTION OPERATION AND PRODUCT CONSTRUCTION

$a_{1,1}$	$a_{1,2}$	$\cdots$	$a_{1,k}$	$a_{1,1}$	$a_{1,2}$	$\cdots$	$a_{1,k}$	$\cdots$	$a_{1,1}$	$a_{1,2}$	$\cdots$	$a_{1,k}$
$a_{2,1}$	$a_{2,2}$	$\cdots$	$a_{2,k}$	$a_{2,1}$	$a_{2,2}$	$\cdots$	$a_{2,k}$	$\cdots$	$a_{2,1}$	$a_{2,2}$	$\cdots$	$a_{2,k}$
$\vdots$			$\vdots$	$\vdots$			$\vdots$	$\cdots$	$\vdots$			$\vdots$
$a_{N,1}$	$a_{N,2}$	$\cdots$	$a_{N,k}$	$a_{N,1}$	$a_{N,2}$	$\cdots$	$a_{N,k}$	$\cdots$	$a_{N,1}$	$a_{N,2}$	$\cdots$	$a_{N,k}$
$b_{1,1}$	$b_{1,1}$	$\cdots$	$b_{1,1}$	$b_{1,2}$	$b_{1,2}$	$\cdots$	$b_{1,2}$	$\cdots$	$b_{1,l}$	$b_{1,l}$	$\cdots$	$b_{1,l}$
$b_{2,1}$	$b_{2,1}$	$\cdots$	$b_{2,1}$	$b_{2,2}$	$b_{2,2}$	$\cdots$	$b_{2,2}$	$\cdots$	$b_{2,l}$	$b_{2,l}$	$\cdots$	$b_{2,l}$
$\vdots$			$\vdots$	$\vdots$			$\vdots$	$\cdots$	$\vdots$			$\vdots$
$b_{M,1}$	$b_{M,1}$	$\cdots$	$b_{M,1}$	$b_{M,2}$	$b_{M,2}$	$\cdots$	$b_{M,2}$	$\cdots$	$b_{M,l}$	$b_{M,l}$	$\cdots$	$b_{M,l}$

Figure 2.1: The Product of A with B

2.5 Compatibility of the Projection Operation and Product Construction

We now cover the conditions needed in order to perform the projection and product constructions together.

2.5.1 Product after Projection

Let C be an $MCA(N; 2, k, (v_1, \dots, v_k))$. Suppose that C has a $m \times s$ projective sub array. From Section 2.2, we use the projection algorithm on C to obtain a new MCA , C' , $MCA(N - m'; 2, k + m', (v_1 - m', \dots, v_k - m')(s')^{m'})$. If there exists an $MCA(M; 2, l, v_k - m')$, then we can apply the product construction from section 2.4.

Thus the only conditions need to project first and then product is the existence of a $MCA(M; 2, l, v_k - m')$. If C was an $MCA(N; 2, k, v_k)$, then the condition would be already satisfied from the corollary in section 2.4. In

2.5. COMPATIBILITY OF THE PROJECTION OPERATION AND PRODUCT CONSTRUCTION

any case, such an array always exists for some M .

2.5.2 Projection after Product

Let C be an $MCA(N; 2, k, (v_1, \dots, v_k))$ that has a $m \times s$ projective sub-array and $v_1 \leq \dots \leq v_k$. Perform the product algorithm in section 2.4 on C with a $MCA(M; 2, l, v_k)$. We obtain a new MCA , C' , $MCA(N + M - 1; 2, kl, (v_1, \dots, v_k)^l)$. We check the conditions for C' to still have a $m \times s$ projective sub-array.

By the product algorithm we know that the first N rows and k columns in C' are an exact copy of C so the $m \times s$ projective sub-array is still there. We only need to consider if it is still private given the sub-array formed from the $N + M - 1$ rows in C' and the first k columns in C' .

Below a copy of C , from the product algorithm we know that the new $M - 1$ rows are constant rows. Let i and j be two columns from the projective sub-array. Let σ_i and σ_j be symbols in columns i and j respectively. Suppose σ_i and σ_j were a private pair in C . Since the new rows are all constant, then the only case where σ_i and σ_j will not be a private pair in C' is if they are equal. Thus we require that the rows of the projective sub-array are also Latin. This can always be achieved:

Lemma 2.12. *Let C be an $MCA(N; 2, k, (v_1, \dots, v_k))$ that has a $m \times s$ projective sub-array. Then suitable symbol permutations can be found for the s columns so that each private pair in the $m \times s$ projective sub-array will remain private after performing the product construction.*

Proof. From the algorithm in 2.2, we only require the pairs to remain private

2.6. PROJECT OR PRODUCT FIRST, THAT IS THE QUESTION

in the s' chosen columns so we only need to consider the $m \times s'$ array. Since $1 \leq m' \leq m \leq v_1$ and $1 \leq s' \leq m - m' \leq v_1$ we get that $s' \leq v_1$. Because each of the s' columns in the projective sub-array are Latin and $s' \leq v_1$, we can always relabel the symbols in the s' columns to build Latin rows for the $m \times s'$ array. $\square$

2.6 Project or Product First, That is the Question

The use of disjoint rows becomes useful when we are looking to optimize constructions. Given a set of disjoint rows, we are able to permute alphabets so that with the product construction we create identical rows thus allowing us to remove them and still have an *MCA*.

But what happens when we do not know anything about the disjoint rows in advance? We do know there can only ever be at most $v_1 = \min\{v_1, \dots, v_k\}$ of these rows. If we project, are these rows still disjoint after projection or are any of them removed by projection? Without knowing where these rows are and if they are a part of the projective sub-array then there is very little we can say about them in full generality. So, if we don't know anything about them in advance what is our best choice: project or product first? We examine cases for extreme sets of disjoint rows.

When comparing two covering arrays, if they both have the same number of rows and one has more columns, that is clearly the better of the two since you will always be able to go from the one with more columns to the one with less by remove those extra columns. Similarly, if both have the same number of columns, then the one with fewer rows would clearly be better as

2.6. PROJECT OR PRODUCT FIRST, THAT IS THE QUESTION

it would require less tests in our testing example. By extension a covering array that has fewer rows and more columns is also better.

In all cases we assume the required $m \times s$ projective sub-array exists.

Case 1: We have two *MCA*s each having only one disjoint row

$$A \text{ is an } MCA(N; 2, k, (v_1, \dots, v_k) : 1)$$

$$B \text{ is an } MCA(M; 2, l, v_k : 1)$$

Take the Product then Project:

$$A \times B \text{ is an } MCA(N + M - 1; 2, kl, (v_1, \dots, v_k)^l : 1)$$

$$P_m(A \times B) \text{ is an } MCA(N + M - 1 - m; 2, kl + m, (v_1 - m, \dots, v_k - m)^l s^m : 1)$$

Project then take the Product:

$$P_m(A) \text{ is an } MCA(N - m; 2, k + m, (v_1 - m, \dots, v_k - m)s^m : 1)$$

$$P_m(B) \text{ is an } MCA(M - m; 2, l + m, (v_1 - m, \dots, v_k - m)s^m : 1)$$

$$P_m(A) \times P_m(B)$$

$$\text{is an } MCA(M + N - 2m - 1; 2, (l + m)(k + m), ((v_1 - m, \dots, v_k - m)s^m)^{l + m} : 1)$$

Comparison:

	Rows	Columns
Product to Project	$N + M - 1 - m$	$kl + m$
Project to Product	$N + M - 1 - 2m$	$(k + m)(l + m)$

Conclusion:

When we project first and then take the product we end up with an *MCA*

2.6. PROJECT OR PRODUCT FIRST, THAT IS THE QUESTION

with fewer rows and more columns, which is obviously better.

Case 2: We have two *MCA*s each having v_1 disjoint rows

A is an $MCA(N; 2, k, (v_1, \dots, v_k) : v_1)$

B is an $MCA(M; 2, l, v_k : v_1)$

Take the Product then Project:

$A \times B$ is an $MCA(N + M - v_1; 2, kl, (v_1, \dots, v_k)^l : v_1)$

$P_m(A \times B)$ is an $MCA(N + M - v_1 - m; 2, kl + m, (v_1 - m, \dots, v_k - m)^l s^m : v_1 - m)$

Project then take the Product:

$P_m(A)$ is an $MCA(N - m; 2, k + m, (v_1 - m, \dots, v_k - m) s^m : v_1 - m)$

$P_m(B)$ is an $MCA(M - m; 2, l + m, (v_1 - m, \dots, v_k - m) s^m : v_1 - m)$

$P_m(A) \times P_m(B)$

is an $MCA(M + N - m - v_1; 2, (l + m)(k + m), ((v_1 - m, \dots, v_k - m) s^m)^{l+m} : v_1 - m)$

Comparison:

	Rows	Columns
Product to Project	$N + M - v_1 - m$	$kl + m$
Project to Product	$N + M - v_1 - m$	$(k + m)(l + m)$

Conclusion:

When we project first and then take the product we end up with an *MCA*

2.6. PROJECT OR PRODUCT FIRST, THAT IS THE QUESTION

with the same number of rows but more columns which is again better.

Case 3: We have two *MCA*s one with one disjoint row and the other with v_1 independent rows

$$A \text{ is an } MCA(N; 2, k, (v_1, \dots, v_k) : v_1)$$

$$B \text{ is an } MCA(M; 2, l, v_k : 1)$$

Take the Product then Project:

$$A \times B \text{ is an } MCA(N + M - 1; 2, kl, (v_1, \dots, v_k)^l : v_1)$$

$$P_m(A \times B) \text{ is an } MCA(N + M - 1 - m; 2, kl + m, (v_1 - m, \dots, v_k - m)^l s^m : v_1 - m)$$

Project then take the Product:

$$P_m(A) \text{ is an } MCA(N - m; 2, k + m, (v_1 - m, \dots, v_k - m) s^m : v_1 - m)$$

$$P_m(B) \text{ is an } MCA(M - m; 2, l + m, (v_1 - m, \dots, v_k - m) s^m : 1)$$

$$P_m(A) \times P_m(B)$$

$$\text{is an } MCA(M + N - 2m - 1; 2, (l + m)(k + m), ((v_1 - m, \dots, v_k - m) s^m)^{l+m} : v_1)$$

Comparison:

	Rows	Columns
Product to Project	$N + M - 1 - m$	$kl + m$
Project to Product	$N + M - 1 - 2m$	$(k + m)(l + m)$

2.6. PROJECT OR PRODUCT FIRST, THAT IS THE QUESTION

Conclusion:

When we project first and then take the product we end up with an *MCA* with fewer rows and more columns. As in case 1, this is always better.

Case 4: We have two *MCA*s one with v_i disjoint rows and the other with one independent row

A is an $MCA(N; 2, k, (v_1, \dots, v_k) : 1)$

B is an $MCA(M; 2, l, v_k : v_1)$

Take the Product then Project:

$A \times B$ is an $MCA(N + M - v_1; 2, kl, (v_1, \dots, v_k)^l : 1)$

$P_m(A \times B)$ is an $MCA(N + M - v_1 - m; 2, kl + m, (v_1 - m, \dots, v_k - m)^l s^m : 1)$

Project then take the Product:

$P_m(A)$ is an $MCA(N - m; 2, k + m, (v_1 - m, \dots, v_k - m)s^m : 1)$

$P_m(B)$ is an $MCA(M - m; 2, l + m, (v_1 - m, \dots, v_k - m)s^m : v_1 - m)$

$P_m(A) \times P_m(B)$

is an $MCA(M + N - m - v_1; 2, (l + m)(k + m), ((v_1 - m, \dots, v_k - m)s^m)^{l + m} : 1)$

Comparison:

	Rows	Columns
Product to Project	$N + M - v_1 - m$	$kl + m$
Project to Product	$N + M - v_1 - m$	$(k + m)(l + m)$

Conclusion:

When we project first and then take the product we end up with an *MCA* with the same number of rows but more columns. As with case 2, this is again better.

Lemma 2.13. *Suppose there exists any two combinations of the following MCAs:*

1. $MCA(N; 2, k, (v_1, \dots, v_k) : 1)$
2. $MCA(N; 2, k, (v_1, \dots, v_k) : v_1)$

Then projection before the product construction will produce an MCA with at most the same number of rows and at least the same number of columns as taking the product before projecting.

We have shown that it is always better to project first and then product second when we have any combination of 1 and v_1 disjoint rows. We conjecture that this holds true in general for any arbitrary set of disjoint rows.

Conjecture 2.14. *Taking the product construction after the projection construction will always be better than taking the product before the projection.*

2.7 Measure of Goodness of CA

In the previous section we could easily determine the better of the two *CAs*. The parameters fell into one of the following two cases:

1. One had less rows and more columns

2.7. MEASURE OF GOODNESS OF CA

2. Both had the same number of rows but one had more columns

If we do not have this nice property, how should we compare two CA s with the same v ?

In this section we construct a method to measure the “goodness” of a covering array which gives us a way to compare two covering arrays with the same alphabet size but with different N ’s and k ’s. This is really the “goodness” of a particular CA ’s use in the product construction. We focus on the product construction because it is the only general recursive construction for increasing k with a fixed v .

Suppose a $CA(N; 2, k, v)$ and a $CA(M; 2, l, v)$ both exist. If asked to construct a covering array of alphabet size v and number of factors g where l and k are both less than g , we could take either covering array and product it with itself until we reached our desired size. The question then becomes which covering array should we choose? Will one give us the same number of factors but with less rows?

We begin by taking both covering arrays and taking the product them with themselves. These two sequences of products yields the following covering arrays:

$$\begin{array}{ll}
 CA(N; 2, k, v) & CA(M; 2, l, v) \\
 CA(2N - 1; 2, k^2, v) & CA(2M - 1; 2, l^2, v) \\
 CA(3N - 2; 2, k^3, v) & CA(3M - 2; 2, l^3, v) \\
 \vdots & \vdots \\
 CA(sN - (s - 1); 2, k^s, v) & CA(aM - (a - 1); 2, l^a, v)
 \end{array}$$

Suppose now that $k^s = l^a$ (it should be noted at this point that $k^s = l^a$ is unlikely to happen but we will assume continuity to derive a useful measure).

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

Then $s \log(k) = a \log(l)$.

$$\begin{aligned}
 sN - (s - 1) &> aM - (a - 1) \\
 \iff sN - s &> aM - a \\
 \iff s(N - 1) &> a(M - 1) \\
 \iff \frac{a \log(l)(N-1)}{\log(k)} &> \frac{s \log(k)(M-1)}{\log(l)} \\
 \iff \frac{a \log(l)(N-1)}{a \log(l) \log(k)} &> \frac{s \log(k)(M-1)}{s \log(k) \log(l)} \\
 \iff \frac{N-1}{\log(k)} &> \frac{M-1}{\log(l)}
 \end{aligned}$$

Definition 2.15. Let C be a $CA(N; 2, k, v)$ then the *goodness* of C is the value of $\frac{N-1}{\log(k)}$.

This measure obviously agrees with our previous measure of two CAs when either N or k was fixed or they were different in opposite directions.

2.8 Construction Strategies and Comparisons

In the previous sections we have shown that it is better to project before using the product construction. We also have a method to measure the goodness of an MCA using only the parameters N and k .

In this section we compare different strategies for choosing the prime to construct an OAF s combined with different combinations of the product construction. We then define the construction of OAM s.

The idea behind these constructions is that when looking to construct an MCA with parameters $v_1 \leq \dots \leq v_k$ and k when $t = 2$, we can either:

- (1) Using *MOLS*, build an OA with $v = v_k$, $k' = \text{number of } MOLS + 2$ and then product to the right k . We then remove symbols from the columns

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

so that the v_i 's match; or

(2) If v_k is not a prime power, we could go up to the next prime power and build an $OAFF$, project to the right v_k , product until the right k and remove symbols from the columns so that the v_i 's match.

From the measure developed in Section 7, we can determine which of these two methods will produce a better MCA without the need to actually use the product construction. Thus, we are only looking to compare the OAM construction with the best $OAFF$ construction followed by the projection construction.

2.8.1 Up to the Next Prime Power

It is well known that for all orders of $q = p^\alpha$, where p is a prime, projective planes exist and their existence is equivalent to optimal CAs .

Theorem 2.16. *Let $q = p^\alpha$ be a prime power. Then there exists a $CA(q^2; 2, q+1, q : 1)$ and $CA(q^2; 2, q, q : q)$.*

Proof. We will be using the construction found in [8] to construct the CAs .

Let $i, j, l \in GF(q)$ and q be a prime power. Then C is an $q^2 \times (q+1)$ covering array defined by $C_{iq+j,l} = i \times l + j$ and for the last column: $C_{iq+j,q} = i$.

To produce the $CA(q^2; 2, q, q : q)$, remove the $(q+1)$ th column from the above definition.

□

Example 2.17. *The following example illustrates this construction for $q = 5$ where $GF(5) = \mathbb{Z}_5$. If we remove the $(q+1)$ th column, the first 5 rows are*

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

disjoint.

When given a v that is not a prime power we can always increase v until it is a prime power, say q , and then project down to obtain the correct v . Applying the projection construction on $CA(q^2; 2, q+1, q : 1)$ and $CA(q^2; 2, q, q : q)$ will produce a $CA(q^2 - (q - v); 2, q + 1 + (q - v), q - (q - v) = v : 1)$ and $CA(q^2 - (q - v); 2, q + (q - v), q - (q - v) = v : q - v)$.

The goodness of a $CA(q^2 - q + v; 2, 2q + 1 - v), v : 1)$ is:

$$\frac{q^2 - q + v - 1}{\log(2q + 1 - v)}.$$

The goodness of a $CA(q^2 - q + v; 2, 2q - v, v : q - v)$ is:

$$\frac{q^2 - q + v - 1}{\log(2q - v)}.$$

2.8.2 Up Two Prime Powers

In the previous sections we stated that given some v that is not a prime power, we can construct a $CA(N; 2, k, v)$ by going up to the next prime power and projecting down. Could we construct a better CA if we went up to the subsequent prime power?

In this section we will show that for any v , we will do best by only going to the next highest prime power.

Theorem 2.18. *For all positive integers v, p and q such that p and q are prime powers and $v \leq p < q$, Then*

$$\frac{p^2 - p + v - 1}{\log(2p + 1 - v)} < \frac{q^2 - q + v - 1}{\log(2q + 1 - v)}$$

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

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

Figure 2.2: $OA(5^2; 2, 6, 5)$ for $q = 5$

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

Proof. Let

$$f_v(p) = \frac{p^2 - p + v - 1}{\log(2p + 1 - v)} \quad \text{for all } p \geq v \geq 1$$

First we show that, $f_v(p)$ is an increasing function of p in this range $1 \leq v \leq p$.

$$\begin{aligned} \frac{\partial f_v(p)}{\partial p} &= \frac{\ln(2)(2p-1)}{\ln(2p+1-v)} - \frac{2(p^2-p+v-1)\ln(2)}{[\ln(2p+1-v)]^2(2p+1-v)} > 0 \\ \iff \frac{\ln(2)(2p-1)}{\ln(2p+1-v)} &> \frac{2(p^2-p+v-1)\ln(2)}{[\ln(2p+1-v)]^2(2p+1-v)} \\ \iff (2p-1)(2p+1-v) &> \frac{2(p^2-p+v-1)}{\ln(2p+1-v)} \\ \iff \frac{4p^2+2p-2pv-2p-1+v}{2p^2-2p+2v-2} &> \frac{1}{\ln(2p+1-v)} \\ \iff \frac{4p^2-2pv-1+v}{2p^2-2p+2v-2} - \frac{1}{\ln(2p+1-v)} &> 0 \end{aligned}$$

For fixed p , for all $v \leq p$, $\frac{4p^2-2pv-1+v}{2p^2-2p+2v-2} - \frac{1}{\ln(2p+1-v)}$ is always decreasing as a function of v . Then,

$$\begin{aligned} \frac{4p^2-2pv-1+v}{2p^2-2p+2v-2} - \frac{1}{\ln(2p+1-v)} &> \frac{4p^2-2p(v+1)-1+(v+1)}{2p^2-2p+2(v+1)-2} \\ &\quad - \frac{1}{\ln(2p+1-(v+1))} \\ \iff \frac{4p^2-2pv-1+v}{2p^2-2p+2v-2} - \frac{4p^2-2pv-2p+v}{2p^2-2p+2v} &> \frac{1}{\ln(2p+1-v)} - \frac{1}{\ln(2p-v)} \\ \iff \frac{p(p+1)(2p-1)}{2(p^2-p+v-1)(p^2-p+v)} &> \frac{\ln(2p-v)-\ln(2p+1-v)}{\ln(2p+1-v)\ln(2p-v)} \end{aligned}$$

since $\ln(2p+1-v)\ln(2p-v) > 0$ and $\ln(2p-v) < \ln(2p-v+1)$

$$\frac{\ln(2p-v)-\ln(2p+1-v)}{\ln(2p+1-v)\ln(2p-v)} < 0.$$

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

Since $p \geq v \geq 1$,

$$\frac{p(p+1)(2p-1)}{2(p^2-p+v-1)(p^2-p+v)} > 0.$$

Hence for a fixed p , $f'_v(p)$ is decreasing for all $v \leq p$.

Since

$$\frac{\partial f_v(p)}{\partial p} = \frac{\ln(2)(2p-1)}{\ln(2p+1+v)} - \frac{2(p^2-p+v-1)\ln(2)}{[\ln(2p+1-v)]^2(2p+1-v)}$$

is at it's smallest when $v = p$, fix $v = p$. Then,

$$\begin{aligned} \frac{4p^2 - 2p^2 - 1 + p}{2p^2 - 2p + 2p - 2} - \frac{1}{\ln(2p+1-p)} &> 0 \\ \frac{2p^2 + p - 1}{2p^2 - 2} &> \frac{1}{\ln(p+1)} \\ \ln(p+1)(2p^2 + p - 1) &> 2p^2 - 2 \end{aligned}$$

for all $p \geq 1$. Hence $f_v(p)$ is increasing for all $p \geq v \geq 1$.

By hypothesis, since $q > p$ we get

$$\frac{q^2 - q + v - 1}{\log(2q + 1 - v)} > \frac{p^2 - p + v - 1}{\log(2p + 1 - v)}.$$

□

What we can conclude from this is that for any v that is not a prime power, we never need to go past the first closest prime power to obtain a CA with best “goodness”.

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

2.8.3 OAFFs with Disjoint Rows

In Section 2.6 we conjectured that if the values and placement for the set of disjoint rows are unknown in advance then our best combination is to project first and then product and we proved it for extreme sets of disjoint rows. In section 2.8.1 we gave the construction for an *OAFF* that let us know where the disjoint rows were:

The *MCA* with 1 disjoint row is

$$(1) \text{ is a } MCA(q^2; 2, q + 1, q : 1)$$

The *MCA* with q disjoint rows is

$$(2) \text{ is a } MCA(q^2; 2, q, q : q)$$

Let $P_m(1)$ and $P_m(2)$ be the two *MCAs* after we perform the projection algorithm m times from Theorem 2.6. In this section we will determine which combinations of $P_m(1)$ and $P_m(2)$ with the product construction will give us the best *MCA* with 1 disjoint row and $q - m$ disjoint rows (these are precisely the cases from Section 2.6 where we know it is always best to project first). Let $P_m(1)$ and $P_m(2)$ be defined as follows:

$$P_m(1) \text{ is a } MCA(q^2 - m; 2, q + 1 + m, q - m : 1)$$

$$P_m(2) \text{ is a } MCA(q^2 - m; 2, q + m, q - m : q - m)$$

We now apply the product construction on each of the four possible combinations of $P_m(1)$ and $P_m(2)$.

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

$P_m(1) \times P_m(1)$ is a $MCA(2q^2 - 2m - 1; 2, (q + 1 + m)^2, q - m : 1)$

$P_m(1) \times P_m(2)$ is a $MCA(2q^2 - m - q; 2, (q + 1 + m)(q + m), q - m : 1)$

$P_m(2) \times P_m(1)$ is a $MCA(2q^2 - 2m - 1; 2, (q + 1 + m)(q + m), q - m : q - m)$

$P_m(2) \times P_m(2)$ is a $MCA(2q^2 - m - q; 2, (q + m)^2, q - m : q - m)$

The next two theorems will prove that when we want to construct an MCA with one disjoint row, our best option is to use $P_m(1) \times P_m(2)$ rather than $P_m(1) \times P_m(1)$. If we want to construct an MCA with $q - m$ disjoint rows, our best options is to use $P_m(2) \times P_m(2)$ rather than $P_m(2) \times P_m(1)$.

To prove this we will need the following:

Lemma 2.19. $(1 + \frac{1}{n})^n$ increases monotonically.

Proof. Let $T_n = (1 + \frac{1}{n})^n$. By the binomial theorem

$$\begin{aligned} T_n &= 1 + n \frac{1}{n} + \frac{n(n-1)}{2!} \frac{1}{n^2} + \cdots + \frac{n(n-1) \cdots 1}{n!} \frac{1}{n^n} \\ &= 1 + 1 + (1 - \frac{1}{n}) \frac{1}{2!} + \cdots + (1 - \frac{1}{n})(1 - \frac{2}{n}) \cdots (1 - \frac{n-1}{n}) \frac{1}{n!} \end{aligned}$$

By replacing n with $n + 1$ we cause each term to increase and add another positive term, thus T_n increases monotonically. $\square$

The following theorem shows that if we want to end up with an MCA with only one disjoint row then our best option is to product an MCA with one disjoint row with an MCA with $q - m$ disjoint rows; i.e. $P_m(1) \times P_m(2)$ rather than $P_m(1) \times P_m(1)$.

Bertrand's postulate tells us that for any x , there will be a prime at most $2x$ distance away [2]. Since we have shown that we never need to go any higher then the next highest prime and that prime is at most $2x$ distance away, we only need to focus on the interval $0 \leq m \leq (q/2) - 1$.

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

Theorem 2.20. Let $f_m(q) = \frac{2q^2-2m-2}{\log((q+m+1)^2)}$ and $g_m(q) = \frac{2q^2-q-m-1}{\log[(q+m+1)(q+m)]}$ with $0 \leq m \leq (q/2) - 1$ and $q \geq 5$ be the goodness of $P_m(1) \times P_m(1)$ and $P_m(1) \times P_m(2)$ respectively. Then $f_m(q) > g_m(q)$.

Proof. We have $f_m(q) = \frac{2q^2-2m-2}{\log((q+m+1)^2)} > g_m(q) = \frac{2q^2-q-m-1}{\log[(q+m+1)(q+m)]}$

$$\begin{aligned} (2q^2 - 2m - 2)(\log((q + 1 + m)(q + m))) &> (2q^2 - q - m - 1)(\log((q + 1 + m)^2)) &\iff \\ (q + m)^{2q^2-2m-2} &> (q + 1 + m)^{2q^2-2q} &\iff \\ (q^2 - m - 1) \log(q + m) &> (2q^2 - 2q) \log(q + 1 + m) \end{aligned}$$

We first show the desired inequality at the end points of the interval $0 \leq m \leq (q/2) - 1$.

Let $m = 0$.

$$\begin{aligned} (q^2 - 1) \log(q) &> (2q^2 - 2q) \log(q + 1) &\iff \\ (q + 1) \log(q) &> q \log(q + 1) &\iff \\ \frac{(q+1)}{\log(q+1)} &> \frac{q}{\log(q)} \end{aligned}$$

Since $\frac{q}{\log(q)}$ is an increasing function, $\frac{q}{\log(q)} < \frac{q+1}{\log(q+1)}$.

Now let $m = (q/2) - 1$.

$$\begin{aligned} (q^2 - \frac{q}{2}) \log(\frac{3}{2}q - 1) &> (q^2 - q) \log(\frac{3}{2}q) &\iff \\ q(2q - 1) \log(\frac{3}{2}q - 1) &> q(2q - 2) \log(\frac{3}{2}q) &\iff \\ (2q - 1) \log(\frac{3}{2}q - 1) &> (2q - 2) \log(\frac{3}{2}q) &\iff \\ 2q \log\left(\frac{3q-2}{3q}\right) &> \log\left(\frac{6q-4}{9q^2}\right) &\iff \\ \left(1 - \frac{2}{3q}\right)^{2q} &> \left(\frac{6q-4}{9q^2}\right) \end{aligned}$$

Since $(1 - \frac{2}{3q})^{2q}$ is an increasing function, which can easily be seen by taking the derivative with $q \geq 5$, we have $(\frac{13}{15})^{10} \leq (1 - \frac{2}{3q})^{2q}$. This gives $(\frac{13}{15})^{10} > \left(\frac{6q-4}{9q^2}\right) \iff 9(\frac{13}{15})^{10}q^2 - 6q - 6 > 0$. Taking the derivative we get $18(\frac{13}{15})^{10}q - 6 > 0 \iff q > \frac{6}{18}(\frac{15}{13})^{10} \implies q \geq 5$. Thus $\left(1 - \frac{2}{3q}\right)^{2q} > \left(\frac{6q-4}{9q^2}\right)$

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

for all $q \geq 5$.

We have now shown that the desired inequality holds at the end points of the interval. We will now show that for a fixed q ,

$$(q^2 - m - 1) \log(q + m) > (q^2 - q) \log(q + 1 + m) \iff$$

$$\frac{(q^2 - m - 1) \log(q + m)}{(q^2 - q) \log(q + 1 + m)} \text{ is decreasing on the interval } 0 \leq m \leq \frac{q}{2} - 1.$$

Let q be fixed. We have $\frac{(q^2 - m - 1) \log(q + m)}{(q^2 - q) \log(q + 1 + m)}$ is decreasing on the interval $0 \leq m \leq \frac{q}{2} - 1 \iff \frac{(q^2 - m - 1) \log(q + m)}{\log(q + 1 + m)}$ is decreasing on the interval $0 \leq m \leq \frac{q}{2} - 1$. Let $h(m) = \frac{(q^2 - m - 1) \log(q + m)}{\log(q + 1 + m)}$.

$$\begin{aligned} h'(m) &= \frac{q^2 - m - 1}{(q + m) \log(q + 1 + m)} - \frac{\log(q + m)}{\log(q + 1 + m)} - \frac{(q^2 - m - 1) \log(q + m)}{(\log(q + 1 + m))^2 (q + 1 + m)} < 0 & \iff \\ &\frac{q^2 - m - 1}{(q + m) \log(q + 1 + m)} - \frac{(q^2 - m - 1) \log(q + m)}{(\log(q + 1 + m))^2 (q + 1 + m)} < \frac{\log(q + m)}{\log(q + 1 + m)} & \iff \\ &\frac{q^2 - m - 1}{(q + m)} - \frac{(q^2 - m - 1) \log(q + m)}{\log(q + 1 + m)(q + 1 + m)} < \log(q + m) & \iff \\ \frac{q^2 - m - 1}{\log(q + 1 + m)(q + m)(q + 1 + m)} [\log(q + 1 + m)(q + 1 + m) - \log(q + m)(q + m)] &< \log(q + m) & \iff \\ \frac{q^2 - m - 1}{\log(q + 1 + m)(q + m)(q + 1 + m)} [(q + m) \log(1 + \frac{1}{q + m}) + \log(q + 1 + m)] &< \log(q + m) & \iff \\ \frac{q^2 - m - 1}{(q + m)(q + 1 + m)} \left[\frac{(q + m) \log(1 + \frac{1}{q + m})}{\log(q + 1 + m)} + 1 \right] &< \log(q + m) \end{aligned}$$

By the Lemma 2.18, $(q + m) \log(1 + \frac{1}{q + m})$ increases monotonically and is bounded by $\log(e) = 1$ hence $(q + m) \log(1 + \frac{1}{q + m}) \leq 1$. We get that

$$\frac{q^2 - m - 1}{(q + m)(q + 1 + m)} \leq 1,$$

and

$$\left[\frac{(q + m) \log(1 + \frac{1}{q + m})}{\log(q + 1 + m)} + 1 \right] \leq \left[\frac{1}{\log(q + 1 + m)} + 1 \right] \leq 2.$$

Hence,

$$\frac{q^2 - m - 1}{(q + m)(q + 1 + m)} \left[\frac{(q + m) \log(1 + \frac{1}{q + m})}{\log(q + 1 + m)} + 1 \right] \leq 2 < \log(q + m)$$

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

for $q \geq 5$. Thus $f_m(q) = \frac{2q^2-2m-2}{\log((q+m+1)^2)} > g_m(q) = \frac{2q^2-q-m-1}{\log[(q+m+1)(q+m)]}$ for all $q \geq 5$.

□

The following theorem shows that if we want to end up with an *MCA* with more than one disjoint row than our best option is to product an *MCA* with $q-m$ disjoint rows with an *MCA* with $q-m$ disjoint rows; i.e. $P_m(2) \times P_m(2)$ rather than $P_m(2) \times P_m(1)$.

Theorem 2.21. *Let $f_m(q) = \frac{2q^2-2m-2}{\log[(q+m)(q+m+1)]}$ and $g_m(q) = \frac{2q^2-m-q-1}{2\log(q+m)}$ with $q \geq 5$ and $0 \leq m \leq (q/2) - 1$. Then $f_m(q) > g_m(q)$.*

Proof. We have $f_m(q) = \frac{2q^2-2m-2}{\log[(q+m)(q+m+1)]} > g_m(q) = \frac{2q^2-m-q-1}{\log((q+m)^2)}$

$$\begin{aligned} (2q^2 - 2m - 2)(\log((q+m)^2)) &> (2q^2 - m - q - 1)(\log[(q+m)(q+m+1)]) && \iff \\ (q+m)^{2q^2-3m+q-3} &> (q+1+m)^{2q^2-q-m-1} && \iff \\ (2q^2 - 3m + q - 3)\log(q+m) &> (2q^2 - q - m - 1)\log(q+1+m) \end{aligned}$$

For $0 \leq m \leq (q/2) - 1$, $2q^2 - q - 1 > 2q^2 - q - m - 1$, so if we can show that $(2q^2 - 3m + q - 3)\log(q+m) > (2q^2 - q - 1)\log(q+1+m)$ then we show that $(2q^2 - 3m + q - 3)\log(q+m) > (2q^2 - q - m - 1)\log(q+1+m)$.

We first show the desired inequality at the end points of the interval $0 \leq m \leq (q/2) - 1$.

Let $m = 0$

$$\begin{aligned} (2q^2 + q - 3)\log(q) &> (2q^2 - q - 1)\log(q+1) && \iff \\ (2q+3)\log(q) &> (2q+1)\log(q+1) && \iff \\ \frac{(2q+3)}{\log(q+1)} &> \frac{2q+1}{\log(q)} \end{aligned}$$

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

Since $\frac{2q+1}{\log(q)}$ is an increasing function, $\frac{2q+1}{\log(q)} < \frac{2q+3}{\log(q+1)}$.

Since, $2q^2 - q > 2q^2 - q - 1$, if we can show that $(2q^2 - 3m + q - 3) \log(q+1) > (2q^2 - q) \log(q+1+m)$ then we show that $(2q^2 - 3m + q - 3) \log(q+1) > (2q^2 - q - m - 1) \log(q+1+m)$.

Now let $m = (q/2) - 1$.

$$\begin{aligned} (2q^2 - \frac{q}{2}) \log(\frac{3}{2}q - 1) &> (2q^2 - q) \log(\frac{3}{2}q) &&\iff \\ q(4q - 1) \log(\frac{3}{2}q - 1) &> q(4q - 2) \log(\frac{3}{2}q) &&\iff \\ 4q \log\left(\frac{3q-2}{3q}\right) &> \log\left(\frac{6q-4}{9q^2}\right) &&\iff \\ \left(1 - \frac{2}{3q}\right)^{4q} &> \left(\frac{6q-4}{9q^2}\right) \end{aligned}$$

Since $(1 - \frac{2}{3q})^{4q}$ is an increasing function, which can easily be seen by taking the derivative with $q \geq 5$, we have $(\frac{13}{15})^{20} \leq (1 - \frac{2}{3q})^{2q}$. Since $(\frac{13}{15})^{20} > \left(\frac{6q-4}{9q^2}\right) \iff 9(\frac{13}{15})^{20}q^2 - 6q - 6 > 0$. Taking the derivative we get $18(\frac{13}{15})^{20}q - 6 > 0 \iff q > \frac{6}{18}(\frac{15}{13})^{20} < 5$. Thus $\left(1 - \frac{2}{3q}\right)^{2q} > \left(\frac{6q-4}{9q^2}\right)$ for all $q \geq 5$.

We have now shown that the desired inequality holds at the end points of the interval. We will now show that for a fixed q ,

$$(2q^2 - 3m + q - 3) \log(q+m) > (2q^2 - q) \log(q+1+m) \iff$$

$$\frac{(2q^2 - 3m + q - 3) \log(q+m)}{(2q^2 - q) \log(q+1+m)} \text{ is decreasing on the interval } 0 \leq m \leq \frac{q}{2} - 1.$$

Let q be fixed. Show $\frac{(2q^2 - 3m + q - 3) \log(q+m)}{(2q^2 - q) \log(q+1+m)}$ is decreasing on the interval $0 \leq m \leq \frac{q}{2} - 1 \iff \frac{(2q^2 - 3m + q - 3) \log(q+m)}{\log(q+1+m)}$ is decreasing on the interval $0 \leq m \leq \frac{q}{2} - 1$. Let $h(m) = \frac{(2q^2 - 3m + q - 3) \log(q+m)}{\log(q+1+m)}$.

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

$$\begin{aligned}
h'(m) &= \frac{2q^2-3m+q-3}{(q+m)\log(q+1+m)} - \frac{3\log(q+m)}{\log(q+m+1)} - \frac{(2q^2-3m+q-3)\log(q+m)}{(\log(q+1+m))^2(q+1+m)} < 0 && \iff \\
&\frac{2q^2-3m+q-3}{(q+m)\log(q+1+m)} - \frac{(2q^2-3m+q-3)\log(q+m)}{(\log(q+1+m))^2(q+1+m)} < \frac{3\log(q+m)}{\log(q+m+1)} && \iff \\
&\frac{2q^2-3m+q-3}{(q+m)} - \frac{(2q^2-3m+q-3)\log(q+m)}{\log(q+1+m)(q+1+m)} < 3\log(q+m) && \iff \\
&\frac{2q^2-3m+q-3}{\log(q+1+m)(q+m)(q+m+1)} [\log(q+1+m)(q+1+m) - \log(q+m)(q+m)] < 3\log(q+m) && \iff \\
&\frac{2q^2-3m+q-3}{\log(q+1+m)(q+m)(q+m+1)} [(q+m)\log(1+\frac{1}{q+m}) + \log(q+1+m)] < 3\log(q+m) && \iff \\
&\frac{2q^2-3m+q-3}{(q+m)(q+m+1)} \left[\frac{(q+m)\log(1+\frac{1}{q+m})}{\log(q+1+m)} + 1 \right] < 3\log(q+m)
\end{aligned}$$

By the Lemma 2.18, $(q+m)\log(1+\frac{1}{q+m})$ increases monotonically and is bounded by $\log(e) = 1$ hence $(q+m)\log(1+\frac{1}{q+m}) \leq 1$. We get that

$$\frac{2q^2-3m+q-3}{(q+m)(q+m+1)} \leq 2,$$

and

$$\left[\frac{(q+m)\log(1+\frac{1}{q+m})}{\log(q+1+m)} + 1 \right] \leq \left[\frac{1}{\log(q+1+m)} + 1 \right] \leq 2.$$

Hence,

$$\frac{2q^2-3m+q-3}{(q+m)(q+m+1)} \left[\frac{(q+m)\log(1+\frac{1}{q+m})}{\log(q+1+m)} + 1 \right] \leq 4 < 3\log(q+m).$$

for $q \geq 5$. Thus $f_m(q) = \frac{2q^2-2m-2}{\log((q+m+1)(q+m))} > g_m(q) = \frac{2q^2-q-m-1}{\log((q+m)^2)}$ for all $q \geq 5$. $\square$

Conclusion: From sections 2.8.2, we know that our best *OAFF* is constructed by only going up to the first closest prime power. Once we have projected down to the right v , we now know that to obtain the best *MCA* with only one disjoint row we product $P_m(1)$ with $P_m(2)$ and for the best *MCA* with $q-m$ disjoint rows we product $P_m(2)$ with $P_m(2)$.

In the following section we look at the construction of *OAMs*.

2.8. CONSTRUCTION STRATEGIES AND COMPARISONS

2.8.4 MOLS - Mutually Orthogonal Latin Squares

In this section we look at how to construct *MCAs* from *MOLS*. We begin by covering a couple of basic definitions.

Definition 2.22. *A latin square of side n is an $n \times n$ array in which each cell contains a single symbol from the set 0 to $n - 1$, such that each symbol occurs exactly once in each row and exactly once in each column.*

Definition 2.23. *Two latin squares L and L' of the same order are orthogonal if $L(a, b) = L(c, d)$ and $L'(a, b) = L'(c, d)$, implies $a = c$ and $b = d$.*

Definition 2.24. *A set of latin squares $L_1, \dots, L_m$ is mutually orthogonal, or a set of *MOLS*, if for every $1 \leq i < j \leq m$, L_i and L_j are orthogonal.*

A set of *MOLS* is equivalent to an *OA* and therefor a *CA*.

Theorem 2.25. *Suppose there exists k' *MOLS* of side n . Then there exists an $CA(n^2, 2, k' + 2, n)$.*

Proof. We use the construction in [9].

Let $\{L_i : 1 \leq i \leq k'\}$ be a set of k' *MOLS* on symbols $\{1, \dots, n\}$. Form an $n^2 \times (k' + 2)$ array $A = (a_{ij})$ whose rows are $(i, j, L_1(i, j), L_2(i, j), \dots, L_{k'}(i, j))$ for $1 \leq i, j \leq k'$. □

We have defined two construction strategies to compare *MCAs*, *OAMs* and the best construction for *OAFFs*. From [9] we have tables for the best known number of *MOLS* for side n of values less then 10000 and the latest asymptotic growth function for the number of *MOLS*. We will use the goodness measure to compare the two strategies for values less then 10000

2.9. THE BATTLE OF THE CENTURY - OAM VS OAFF

and then show asymptotically which one is always better for the current asymptotic growth knowledge of *MOLS*.

2.9 The Battle of the Century - OAM vs OAFF

2.9.1 $v < 10000$

Table 3.87 in [9] gives us the best known lower bounds on $N(v)$, the number of *MOLS* of side v for all values of $v < 10000$. Using the values from these tables, we have compared the goodness of a covering array constructed using *MOLS* to one constructed using the next closest prime power.

Definition 2.26. *For any v and $N(v)$, we can always construct an $OA(v^2; 2, N(v) + 2, v)$ from *MOLS*. The goodness of a covering array constructed from *MOLS* is:*

$$\frac{v^2 - 1}{\log(N(v) + 2)}$$

Definition 2.27. *For any v and q (the next closest prime power to v), we can always construct an $OA(q^2; 2, q + 1, q)$. Projecting down to v gives us a $CA(q^2 - (q - v), 2, q + 1 + (q - v), q - (q - v))$. The goodness of a covering array constructed from *OAFF* is:*

$$\frac{q^2 - q + v - 1}{\log(2q + 1 - v)}$$

After a comparison of these two values, from the above definitions, we found that for all $v < 10000$, *OAFF* followed by the projection construction always produces a better *MCA*. The only v where they agree are on prime powers.

2.9. THE BATTLE OF THE CENTURY - OAM VS OAFF

The following section proves this true for sufficiently large values of v given the current best known asymptotic growth of $N(v)$.

2.9.2 Asymptotic Comparison

As we have seen in the last section, for all values of $v < 10000$, constructing *MCAs* from *OAFFs* and then projecting the alphabet sizes down gives us a smaller covering array than those constructed using *OAMs*. In this section we show that as v becomes large enough, the *OAFF* constructions will always be better than *OAM* construction given the current best asymptotic growth for $N(v)$.

Bertrand's postulate states that there is always a prime number between x and $2x$. A recent result, due to Baker, Harman and Pintz, shows that there is always a prime between $x - x^{0.525}$ and x [2]. We will now show that for all $\epsilon > 0$, there is also a prime between x and $x + x^{0.525+\epsilon}$.

Theorem 2.28. *For all $\epsilon > 0$, there exists an x_0 such that for all $x > x_0$, the interval $[x, x + x^{0.525+\epsilon}]$ contains prime numbers.*

Proof. From [2], there exists a prime in the interval $[x - x^{0.525}, x]$ for some $x > x'_0$. We now show that

$$\begin{aligned}
 x &\leq (x + x^{0.525+\epsilon}) - (x + x^{0.525+\epsilon})^{0.525} && \iff \\
 x^{0.525+\epsilon} &\geq (x + x^{0.525+\epsilon})^{0.525} && \iff \\
 x^{1+(\epsilon/0.525)} &\geq x + x^{0.525+\epsilon} && \iff \\
 x^{(\epsilon/0.525)} &\geq 1 + \frac{x^\epsilon}{x^{0.475}}
 \end{aligned}$$

which is true for all $x \geq x_\epsilon$. Letting $x_0 = \max\{x_\epsilon, x'_0\}$ gives the desired result. □

2.9. THE BATTLE OF THE CENTURY - OAM VS OAFF

This gives that for any v we will only ever have to go up to $v + v^{0.525+\epsilon}$ to reach the closest prime to construct an *OAFF* and project $v^{0.525+\epsilon}$ times to construct a *CA* with alphabet size to v .

From [9] we get that for v sufficiently large, $N(v) \geq v^{\frac{1}{14.8}}$. Thus for any large v we can always construct a covering array using *OAMs* that gives us an alphabet size of v with $v^{\frac{1}{14.8}} + 2$ columns.

The goodness of a covering array constructed from *MOLS* is:

$$\frac{v^2 - 1}{\log(N(v) + 2)} = \frac{x^2 - 1}{\log(x^{\frac{1}{14.8}} + 2)}$$

The goodness of a covering array constructed from *OAFF* is:

$$\frac{p^2 - p + v - 1}{\log(2p + 1 - v)} = \frac{(x + x^{0.525+\epsilon})^2 - x^{0.525+\epsilon} - 1}{\log(x + 2x^{0.525+\epsilon} + 1)}$$

Theorem 2.29. For sufficiently large $x = v$,

$$\frac{x^2 - 1}{\log(x^{\frac{1}{14.8}} + 2)} > \frac{(x + x^{0.525+\epsilon})^2 - x^{0.525+\epsilon} - 1}{\log(x + 2x^{0.525+\epsilon} + 1)}$$

Proof.

$$\begin{aligned} \frac{x^2 - 1}{\log(x^{\frac{1}{14.8}} + 2)} &> \frac{(x + x^{0.525+\epsilon})^2 - x^{0.525+\epsilon} - 1}{\log(x + 2x^{0.525+\epsilon} + 1)} && \Longleftrightarrow \\ (x^2 - 1)(\log(x + 2x^{0.525+\epsilon} + 1)) &> ((x + x^{0.525+\epsilon})^2 - x^{0.525+\epsilon} - 1)(\log(x^{\frac{1}{14.8}} + 2)) && \Longleftrightarrow \\ (x^2) &> \frac{(\log(x^{\frac{1}{14.8}} + 2))(2x^{1.050+\epsilon'} - x^{0.525+\epsilon} - 1) + \log(x + 2x^{0.525+\epsilon} + 1)}{\log\left(\frac{x + 2x^{0.525+\epsilon} + 1}{x^{\frac{1}{14.8}} + 2}\right)} \end{aligned}$$

Claim 2.30. $\log(x) > \log(x^{\frac{1}{14.8}} + 2)$ for all $x \geq 4$.

Proof.

$$\begin{aligned} \log(x) &> \log(x^{\frac{1}{14.8}} + 2) && \Longleftrightarrow \\ x &> x^{\frac{1}{14.8}} + 2 && \Longleftrightarrow \\ x - x^{\frac{1}{14.8}} &> 2 && \Longleftrightarrow \\ x(1 - \frac{1}{x^{.9324}}) &> 2 \end{aligned}$$

2.9. THE BATTLE OF THE CENTURY - OAM VS OAFF

Let $f(x) = x(1 - \frac{1}{x^{9324}})$. Since $f(x)$ is an increasing function and $f(4) = 2.90175 > 2$ we get $\log(x) > \log(x^{\frac{1}{14.8}} + 2)$. $\square$

Claim 2.31. $\log(x^2) > \log(x + 2x^{0.525+\epsilon} + 1)$ for all $x \geq 4$.

Proof. Since $\epsilon > 0$, it is sufficient to show that $\log(x^2) > \log(3x + 1)$ for all $x \geq 4$.

$$\begin{aligned} \log(x^2) &> \log(3x + 1) && \iff \\ x^2 &> 3x + 1 && \iff \\ x^2 - 3x &> 1 \end{aligned}$$

Let $f(x) = x^2 - 3x$, then $f'(x) = 2x - 3$ which is greater than 0 for all $x \geq \frac{3}{2}$. When $x = 4$, $f(4) = 4 > 1$ thus $\log(x^2) > \log(x + 2x^{0.525+\epsilon} + 1)$ for all $x \geq 4$. $\square$

Claim 2.32. $\log\left(\frac{x+2x^{0.525+\epsilon}+1}{x^{\frac{1}{14.8}}+2}\right) > \log(x^{0.525})$ for all $x \geq 1$.

Proof. Since $\epsilon > 0$, it is sufficient to show that $\log\left(\frac{x+2x^{0.525}+1}{x^{\frac{1}{14.8}}+2}\right) > \log(x^{0.525})$ for all $x \geq 1$.

$$\begin{aligned} \log\left(\frac{x+2x^{0.525}+1}{x^{\frac{1}{14.8}}+2}\right) &> \log(x^{0.525}) && \iff \\ \frac{x+2x^{0.525}+1}{x^{\frac{1}{14.8}}+2} &> x^{0.525} && \iff \\ x + 2x^{0.525} + 1 &> x^{0.592} + 2x^{0.525} && \iff \\ x - x^{0.592} &> -1 \end{aligned}$$

Let $f(x) = x - x^{0.592}$. $f'(x) = 1 - \frac{0.592}{x^{0.408}}$. Find the range of x such that $f'(x) > 0$. $x^{0.408} > 0.592$ for all $x \geq 1$. $f(1) = 0 > -1$ hence $\log\left(\frac{x+2x^{0.525+\epsilon}+1}{x^{\frac{1}{14.8}}+2}\right) > \log(x^{0.525})$ for all $x \geq 1$. $\square$

Now we show

2.9. THE BATTLE OF THE CENTURY - OAM VS OAFF

$$\begin{aligned}
 x^2 &> \frac{\log(x)(2x^{1.525+\epsilon} + x^{1.050+\epsilon'} - x^{0.525+\epsilon} - 1) + 2\log(x)}{0.525 \log(x)} \implies \\
 x^2 &> \frac{(\log(x^{14.8} + 2))(2x^{1.525+\epsilon} + x^{1.050+\epsilon'} - x^{0.525+\epsilon} - 1) + \log(x + 2x^{0.525+\epsilon} + 1)}{\log\left(\frac{x + 2x^{0.525+\epsilon} + 1}{x^{(1/14.8)} + 2}\right)}
 \end{aligned}$$

From the previous section, we have shown that for any $\epsilon < 0.475$ the approximations we need always hold and thus we leave out ϵ for the remainder of the proof.

$$\begin{aligned}
 x^2 &> \frac{\log(x)(2x^{1.525} + x^{1.050} - x^{0.525} - 1) + 2\log(x)}{0.525 \log(x)} && \iff \\
 x^2 &> \frac{1}{0.525}(2x^{1.525} + x^{1.050} - x^{0.525} + 1) && \iff \\
 x^2 - \frac{2x^{1.525}}{0.525} - \frac{x^{1.050}}{0.525} + \frac{x^{0.525}}{0.525} &> \frac{1}{0.525}
 \end{aligned}$$

$$\text{Let } f(x) = x^2 - \frac{2x^{1.525}}{0.525} - \frac{x^{1.050}}{0.525} + \frac{x^{0.525}}{0.525}. \quad f'(x) = 2x - \frac{3.05x^{0.525}}{0.525} - 2x^{0.050} + \frac{1}{x^{0.475}}.$$

Find when $f'(x) > 0$.

Let $g(x) = (2x^{0.475} - \frac{3.05}{0.525})$, then $g(x)$ is positive when $2x^{0.475} > \frac{3.05}{0.525} \iff x^{0.475} > \frac{3.05}{1.05}$. This is true for all $x \geq 10$. $g(18) = 2.08424 > 2 \implies g(x) > 2$ for all $x \geq 18$.

For all $x \geq 18$

$$\begin{aligned}
 g(x) &> 2 && \iff \\
 (2x^{0.475} - \frac{3.05}{0.525}) &> 2 && \implies \\
 (2x^{0.475} - \frac{3.05}{0.525}) &> \frac{2}{x^{0.475}} && \iff \\
 x^{0.525}(2x^{0.475} - \frac{3.05}{0.525}) &> 2x^{0.050} && \implies \\
 x^{0.525}(2x^{0.475} - \frac{3.05}{0.525}) &> x^{0.050}(2 - \frac{1}{0.525}) && \iff \\
 2x - \frac{3.05x^{0.525}}{0.525} - 2x^{0.050} + \frac{1}{x^{0.475}} &> 0 && \iff \\
 f'(x) &> 0
 \end{aligned}$$

Hence $f'(x)$ is increasing when $x \geq 18$. $f(21) = 8.24311 > \frac{1}{0.525}$. Thus

$$\frac{x^2 - 1}{\log(x^{14.8} + 2)} > \frac{(x + x^{0.525+\epsilon})^2 - x^{0.525+\epsilon} - 1}{\log(x + 2x^{0.525+\epsilon} + 1)} \text{ for all } x \geq 21. \quad \square$$

For all values of n , constructing *OAFFs* from the closest prime power and then projecting down will always produce a better *MCA* than the *OAM* construction.

2.10 Summary

In this Chapter, we now have derived the conditions to perform the product and multiple projection construction compatibility on *MCAs*. We have shown that when the parameters k and v for $t = 2$ are unknown ahead of time (arbitrary k and v) then the best single approach for constructing *MCAs* is to use a combination of the product construction and the projection construction on *OAFF*. It is always better to project and then to product, and for values of v that are not a prime power, only the closest prime power to v is needed to construct the best *MCA*.

For values of $n \leq 10000$ we have shown that all the time the *OAFF* constructions generate better *MCAs* than the *OAMs*. For v sufficiently large, the *OAFF* construction will always generate a better *MCA*.

*To make an impact, you have
to go to extremes.*

Drive Me Crazy

3

Description of Algorithmic Implementations

3.1 Introduction

In this Chapter, we introduce a software library developed by the IBM Haifa Research Labs called the “Combinatorial Test Services”. This library was designed to generate and manipulate testing input data. We have designed an additional C++ class that can be used on its own or along with IBM’s software library. The motivation for this new class comes from Section 3 where given an *MCA*, we can increase the largest of two alphabets efficiently [26]. We combine this with the method of constructing *MCAs* from *OAFFs* in Chapter 2. In section 4, we give the methods for the C++ class we have created to go with the Combinatorial Test Services.

3.2 CTS

The Combinatorial Test Services (CTS) is a software library for the generation and manipulation of testing input data. A test suit is represented by a table whose columns represent the input parameters to a system under test, and whose rows represent the test configurations or test cases. CTS enables the user to generate small test suites with strong coverage properties, choose regression suites, and perform other useful operations.

In [14], CTS defines the basic classes needed for manipulating combinatorial test cases, test suites, and testing parameters. These classes then provide a large range of services needed by the testing community including:

- Creation of test suites with a given level of coverage
- Creation of test suites of a given maximum size
- Reduction of test suites to manageable regression sets
- Extending an existing test suite to provide a desired level of coverage
- Removal of test cases witch contain forbidden combinations
- Construction of test suites without any forbidden combinations
- Printing the test suites in a format suitable for the use of a test execution engine
- Printing the test suites in a format for import into MS Excel

Recent work on removal of test cases with contain forbidden combinations and construction of test suites without any forbidden combinations can be found in [11] which has just been accepted for publication.

In 2004, this technology had been retired [14].

3.3 Algorithms

In this section, we use the results for constructing *MCA*s when the parameters k and v are unknown ahead of time from Chapter 2 to optimize our construction. We will begin stating a theorem that will increase any one alphabet size in a *MCA*. Then we will show that we can increase the two largest alphabet sizes in a *MCA* very efficiently.

The idea of this algorithm is that if we are looking to build an *MCA* and are given $v_1 \leq v_2 \leq \dots \leq v_k$, then we can build an *OAFF* as before in chapter 2 but this time we will construct it using v_{k-2} and not v_k . If we construct an *OAFF* using v_{k-2} , we can project down to the right alphabet size, repeat the product construction until we achieve the correct k and then increase the two largest alphabet sizes such that all the v_i 's match.

The following theorem specifies how to increase one of the alphabet sizes.

Theorem 3.1. [26] *Let $e \geq 0$, $1 \leq i \leq k$, $v_1 \leq \dots \leq v_k$, and $k' := \max\{j : 1 \leq j \leq k, j \neq i\}$. Suppose there exists an $MCA(N; 2, k, (v_1, \dots, v_k))$, then there exists an $MCA(N + ev_{k'}; 2, k, (v_1, \dots, v_i + e, \dots, v_k))$*

Proof. Let C be an $MCA(N; 2, k, (v_1, \dots, v_k))$. Let $e \geq 0$, $1 \leq i \leq k$, $v_1 \leq \dots \leq v_k$, and $k' := \max\{j : 1 \leq j \leq k, j \neq i\}$. Choose any column i in C . Add $ev_{k'}$ new empty rows to the bottom of C . In column i , add each

new symbol $0 \leq g \leq e$, $v_{k'}$ times in the new rows. In column j , match each symbol x with each symbol g from column i . Match the remaining columns in the same manner.

Since the original N rows in C have gone unchanged and all new symbols have been matched with every symbol y in some column z , we have an $MCA(N + ev_{k'}; 2, k, (v_1, \dots, v_i + e, \dots, v_k))$.

□

The following theorem defines how to increase the two largest alphabet sizes.

Theorem 3.2. [26] *Let $e_1, e_2 \geq 0$, and $v_1 \leq \dots \leq v_k$. Suppose there exists an $MCA(N; 2, k, (v_1, \dots, v_k))$, then there exists an $MCA(N + e_1v_k + e_2v_{k-1} + e_1e_2; 2, k, (v_1, \dots, v_{k-2}, v_{k-1} + e_1, v_k + e_2))$*

Proof. Let C be an $MCA(N; 2, k, (v_1, \dots, v_k))$. Take the second largest alphabet size and using the construction from above create an $MCA(N + e_1v_k; 2, k, (v_1, \dots, v_{k-1} + e_1, v_k))$. Now take the largest alphabet size and using the construction from above and the $MCA(N + e_1v_k; 2, k, (v_1, \dots, v_{k-1} + e_1, v_k))$, we get $MCA(N + e_1v_k + e_2v_{k-1} + e_1e_2; 2, k, (v_1, \dots, v_{k-2}, v_{k-1} + e_1, v_k + e_2))$. □

Note that if the $MCA(N; 2, k, (v_1, \dots, v_k)) = v_{k-1}v_k$, i.e. optimal, then so is the $MCA(N + e_1v_k + e_2v_{k-1} + e_1e_2; 2, k, (v_1, \dots, v_{k-2}, v_{k-1} + e_1, v_k + e_2))$.

In the following section, we give the methods for the class CA that we have created using the above algorithm. This algorithm works with the CTS library from IBM or on it own.

3.4 Documentation

The Constructor Summary and Method Summary only outline how that class operates. The details following both provide their full explanation. (Note: int is short for integer)

Constructor Summary

CA(int [], int)

Constructs an <i>MCA</i> containing the alphabet sizes from the specified collection.

Constructor Detail

CA

public CA(int [], int)

Constructs an <i>MCA</i> using the Algorithm from section 4.3. Requires an array of integers for factor sizes and an integer for the size of array.

3.4. DOCUMENTATION

Method Summary	
int	<code>getK()</code> Returns the parameter <i>k</i> for the <i>MCA</i>
int	<code>getT()</code> Returns the parameter <i>t</i> for the <i>MCA</i>
int	<code>getV()</code> Returns the parameter <i>v</i> for the <i>MCA</i>
int	<code>getN()</code> Returns the parameter <i>N</i> for the <i>MCA</i>
<code>vector< vector< int > ></code>	<code>buildArray(int, int)</code> Builds a vector of vectors and returns the vector
int	<code>getValueAt(int, int)</code> Returns the value at the specified position.
bool	<code>checkCA()</code> Returns True if array is a CA of strength 2.

3.4. DOCUMENTATION

void	<code>printCA()</code> Prints array to the screen.
<code>vector< vector< int > ></code>	<code>buildProfile(int, int)</code> Builds a vector of vectors and returns the vector.
void	<code>removeFromProfile(int, int)</code> Removes an element from the profile vector.
void	<code>printProfile()</code> Prints the profile vector to the screen.

Method Detail

getK

public int getK()

Returns the value of k for an *MCA*.

Returns:

int k for an *MCA*

getT

public int getT()

Returns the value of t for an *MCA*.

Returns:

int t for an *MCA*

getV

public int getV()

Returns the values of v for an *MCA*.

Returns:

int v for an *MCA*

getN

public int getN()

Returns the value of N for an *MCA*.

Returns:

int N for an *MCA*

3.4. DOCUMENTATION

buildArray

```
public vector< vector< int > > buildArray( int r, int c)
```

Builds a vector of vectors with r rows and c columns

Returns:

a vector of vectors with r rows and c columns

getValueAt

```
public int getValueAt( int r, int c)
```

Returns the value in the array at position (r,c) .

Returns:

Returns the symbol in row r and column c

checkCA

```
public bool checkCA()
```

Determines if array is a CA of strength 2.

Returns:

True if array is a CA . False otherwise.

printCA

```
public void printCA()
```

Prints the CA to the screen.

Returns:

Nothing

buildProfile

```
public vector< vector< int > > buildProfile( int v, int k)
```

Builds a vector of vectors with v rows and k columns.

Returns:

Returns a vector of vectors with v rows and k columns

removeFromProfile

```
public void removeFromProfile(int r, int c)
```

Removes symbol c from alphabet v_r .

Returns:

Nothing

printProfile

```
public void printProfile()
```

Prints symbols in each v_i to the screen.

Returns:

Nothing

3.5 Summary

This algorithm is currently being used by Daniel Hoffman (Dept. of Computer Science, University of Victoria) in test generations. Combined with context-free grammars, it has been used in testing of “firewalls for TCP flag attacks, generation of large XML documents and corresponding XPath

3.5. SUMMARY

queries and testing of RSS clients for HTML injection vulnerabilities.” [17].

“We ran your code against our hill climbing implementation on a few cases; yours was much faster.” - Daniel Hoffman

Number of Parameters - $k = 12$

v	Hill Climb (in sec)	OAFF (in sec)
2	0.960	0.055
3	8.460	0.049
4	39.586	0.060
5	124.534	0.064
6	319.934	0.081

Number of Parameters - $k = 23$

v	Hill Climb (in sec)	OAFF (in sec)
2	46.815	0.065
3	466.224	0.088
4	2227.309	0.098
5	> 1800	0.105
6	> 1800	0.154

Our single approach algorithm is only designed to bring values of v up to the next prime. The next step would be to use the Number Theory Library (NTL) in our implementation to make use of the prime powers.

Either you define the moment or the moment defines you.

Tin Cup

4

Conclusions and Future Work

4.1 Summary of Results

The main contribution of this thesis are as follows. For mixed level covering arrays we have generalized the multiple projection construction [8] for orthogonal arrays from finite fields to any mixed level covering array that contains a projective sub-array and the needed symbol coverage in a set of rows (Theorem 2.6). We have shown that applying the product construction [26] to a mixed level covering array that contains a projective sub-array will still contain a projective sub-array after the construction thus allowing one to project after taking the product. We have shown that the product construction will always be available for use after performing the projection construction. Thus allowing one to combine Theorem 2.6 and Theorem 2.10.

The ability to combine the two constructions allows us to choose which to apply first. We have shown that when we have a mixed level covering array with a set of disjoint rows then in the extreme cases, 1 and v_i disjoint

4.1. SUMMARY OF RESULTS

rows, that it is always better to project before taking the product. We have developed a measure of the “goodness” of a covering array that allows a way to compare two covering arrays with the same alphabet size but with different N ’s and k ’s. This again is really the “goodness” of a particular covering array’s utility in the product construction.

For a given v that is not a prime power, one can always go up to the next highest prime and construct an orthogonal array. Using the projection construction we can project down to obtain the correct v . We have shown that we will always construct a better orthogonal array if we use the next highest prime power from v rather than any larger prime power.

Using the construction from Theorem 2.16 for orthogonal arrays allows us to know where and how many disjoint rows we have. Using the measure of “goodness”, we have shown that when we need to construct a mixed level covering array with only one disjoint row then we use Theorem 2.20 and when constructing a mixed level covering array with more than one disjoint row then we use Theorem 2.21.

We have shown by use of empirical data for values of $v < 10000$ that going up to the next prime power, constructing an orthogonal array from finite fields, projecting down and then taking the product will always produce a better covering array than then using mutually orthogonal latin squares to construct an orthogonal array followed by the product construction. Using the best know asymptotic results for mutually orthogonal latin squares, we have shown that this is true for all v asymptotically.

Thus, we have developed the best single approach for constructing a mixed level covering array when the parameters k and v for $t = 2$ of a

covering array are unknown ahead of time (arbitrary k and v). It should be again stressed that it is unrealistic to assume that when constructing a mixed level covering array one would be limited to only a single approach.

4.2 Future Work

Future work on this thesis falls into two categories. The first be the theoretical side. The obvious place to start would be looking at sets of disjoint rows and how the projection construction affects them. If we don't know where they are in the array when we perform the projection construction, are any rows lost from the relabeling process or is any row deleted as it was part of the projective sub-array?

We would also like to prove the conjecture previously stated:

Conjecture 2.13. *Taking the product construction after the projection construction will always be better then taking the product before the projection.*

In [3], Colbourn implemented a deterministic density algorithm for constructing covering arrays. It would be of interest to compare the work done in [3] with the work done in this thesis.

The second category is the applied. Going back to the testing problem, for each test that we run there are two possible outcomes, pass or fail. Problems arise when a test returns a fail. The reasons for this is we cannot be certain which interaction was the cause of the failure. As important as it is to determine an efficient way to test a system, it is just as important to be able to determine which pairs cause failing results. This leads us to another generalization of the Covering Array known as Error Locating Arrays (*ELA*)

[23]. An *ELA* is a covering array that has enough tests with the right overlap so that we can do this kind of analysis to find all the faulty pairs.

Error Locating Arrays are a generalization of covering arrays in which we can guarantee the location and detection of errors whenever the structure of the faulty interactions satisfies certain reasonable assumptions. “The structural information comes from properties of graphs encoding the errors. The faulty interactions can be modeled as edges of a hypergraph (hypergraph is a generalization of a graph, where edges can connect any number of vertices). The *ELA* attempts to discover many “good” pairs and then uses them to find “bad” pairs in a controlled way” [23].

During winter 2010, we will be working with Wurldtech Security Technologies (Vancouver, BC) and MITACS Accelerate Canada to implement *ELA*’s and determine their effectiveness in a real world system. Further research is essential to further generalize the Covering Array so that it may become more serviceable to real world systems. Thus we will also be working with Wurldtech’s design team to determining where our model of the Covering Array does not reflect the real world application accurately. With the help of Dr. Brett Stevens and Dr. Lucia Moura, we will use the information provided by the design team and construct new mathematical generalizations of the Covering Array to meet the current needs of the organization.

This will be a great opportunity to view how our current models compare to real word testing suites that are currently in place. This will also add great insight into expanding the working model so it can be implemented better into their current testing environments.

Bibliography

- [1] R. C. Baker and G. Harman. The difference between consecutive primes. *Proc. London Math. Soc. (3)*, 72(2):261–280, 1996.
- [2] R. C. Baker, G. Harman, and J. Pintz. The difference between consecutive primes. II. *Proc. London Math. Soc. (3)*, 83(3):532–562, 2001.
- [3] Renée C. Bryce and Charles J. Colbourn. The density algorithm for pairwise interaction testing: Research articles. *Softw. Test. Verif. Reliab.*, 17(3):159–182, 2007.
- [4] Ashok K. Chandra, Lawrence T. Kou, George Markowsky, and Shmuel Zaks. On sets of Boolean n -vectors with all k -projections surjective. *Acta Inform.*, 20(1):103–111, 1983.
- [5] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, 1997.
- [6] Myra B. Cohen, Charles J. Colbourn, and Alan C.H. Ling. Constructing strength three covering arrays with augmented annealing. *Discrete Mathematics*, 308(13):2709 – 2722, 2008. Combinatorial Designs: A tribute to Jennifer Seberry on her 60th Birthday.
- [7] Charles J. Colbourn. Combinatorial aspects of covering arrays. *Matematiche (Catania)*, 59(1-2):125–172 (2006), 2004.

- [8] Charles J. Colbourn. Strength two covering arrays: existence tables and projection. *Discrete Math.*, 308(5-6):772–786, 2008.
- [9] Charles J. Colbourn and Jeffrey H. Dinitz, editors. *The CRC Handbook of Combinatorial Designs - Second Edition*. CRC Press, Boca Raton, 2007.
- [10] Charles J. Colbourn, Sosina S. Martirosyan, Gary L. Mullen, Dennis Shasha, George B. Sherwood, and Joseph L. Yucas. Products of mixed covering arrays of strength two. *J. Combin. Des.*, 14(2):124–138, 2006.
- [11] Peter Danziger, Eric Mendelsohn, Lucia Moura, and Brett Stevens. Covering arrays avoiding forbidden edges. to appear *Theoret. Comput. Sci.*, 2009.
- [12] L. Gargano, J. Körner, and U. Vaccaro. Capacities: from information theory to extremal set theory. *J. Comb. Theory Ser. A*, 68(2):296–316, 1994.
- [13] Alan Hartman. Software and hardware testing using combinatorial covering suites. In *Graph theory, combinatorics and algorithms*, pages 237–266. Springer, New York, 2005.
- [14] Alan Hartman and Leonid Raskin. Combinatorial test services - a software package. <http://www.alphaworks.ibm.com/tech/cts>.
- [15] Alan Hartman and Leonid Raskin. Problems and algorithms for covering arrays. *Discrete Math.*, 284(1-3):149–156, 2004.

- [16] A. S. Hedayat, N. J. A. Sloane, and John Stufken. *Orthogonal arrays*. Springer Series in Statistics. Springer-Verlag, New York, 1999. Theory and applications, With a foreword by C. R. Rao.
- [17] Daniel Hoffman, Lewis Sobotkiewicz, Hong-Yi Wang, Paul Strooper, Gary Bazdell, and Brett Stevens. Test generation with context free grammars and covering arrays. to appear IEEE Computer Society digital library, 2009.
- [18] <http://Testcover.com>. Pairwise test case generator, 2009. [Online; accessed 15-August-2009].
- [19] <http://www.testersdesk.com>. Pairwise test case generator, 2009. [Online; accessed 13-August-2009].
- [20] G. O. H. Katona. Two applications (for search theory and truth functions) of sperner type theorems. *Periodica Mathematica Hungarica*, 3(1-2):19–26, 1973.
- [21] Daniel J. Kleitman and Joel Spencer. Families of k -independent sets. *Discrete Math.*, 6:255–262, 1973.
- [22] E. Marczewski. Independance d’ensembles et prolongement de mesures. (résultats et problèmes). *Colloq. Math.*, 1:122–132, 1948.
- [23] Conrado Martinez, Lucia Moura, Daniel Panario, and Brett Stevens. Locating errors using elas, covering arrays and adaptive testing algorithms. to appear SIAM J. Disc. Math., 2009.

- [24] Karen Meagher and Brett Stevens. Covering arrays on graphs. *J. Combin. Theory Ser. B*, 95(1):134–151, 2005.
- [25] Karen Meagher and Brett Stevens. Group construction of covering arrays. *J. Combin. Des.*, 13(1):70–77, 2005.
- [26] Lucia Moura, John Stardom, Brett Stevens, and Alan Williams. Covering arrays with mixed alphabet sizes. *J. Combin. Des.*, 11(6):413–432, 2003.
- [27] Kari J. Nurmela. Upper bounds for covering arrays by tabu search. *Discrete Appl. Math.*, 138(1-2):143–152, 2004. Optimal discrete structures and algorithms (ODSA 2000).
- [28] Alfréd Rényi. *Foundations of probability*. Holden-Day Inc., San Francisco, Calif., 1970.
- [29] N. J. A. Sloane. Covering arrays and intersecting codes. *J. Combin. Des.*, 1(1):51–63, 1993.
- [30] J. Stardom. Metaheuristics and the search for covering and packing arrays. Master’s thesis, Simon Fraser University, 2001.
- [31] Brett Stevens. *Transversal Covers and Packing*. PhD thesis, University of Toronto, 1998.
- [32] Brett Stevens and Eric Mendelsohn. New recursive methods for transversal covers. *J. Combin. Des.*, 7(3):185–203, 1999.
- [33] Jianxing Yin. Constructions of difference covering arrays. *J. Comb. Theory Ser. A*, 104(2):327–339, 2003.