Maximin Strong Orthogonal Arrays

by

Jiaying Weng

B.Sc., Simon Fraser University, 2012

Project Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

in the
Department of Statistics & Actuarial Science
Faculty of Science

© Jiaying Weng 2014

Simon Fraser University
Summer 2014

All rights reserved.
However, in accordance with the Copyright Act of Canada, this work may be
reproduced without authorization under the conditions for “Fair Dealing.”
Therefore, limited reproduction of this work for the purposes of private study,
research, criticism, review and news reporting is likely to be in accordance
with the law, particularly if cited appropriately.
APPROVAL

Name: Jiaying Weng

Degree: Master of Science

Title of Project: Maximin Strong Orthogonal Arrays

Examining Committee: Dr. Tim Swartz
Professor
Chair

Dr. Boxin Tang
Professor
Senior Supervisor

Dr. Jiguo Cao
Associate Professor
Supervisor

Dr. Michelle Zhou
Assistant Professor
Internal Examiner

Date Approved: August 18th, 2014
Partial Copyright Licence

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the non-exclusive, royalty-free right to include a digital copy of this thesis, project or extended essay[s] and associated supplemental files ("Work") (title[s] below) in Summit, the Institutional Research Repository at SFU. SFU may also make copies of the Work for purposes of a scholarly or research nature; for users of the SFU Library; or in response to a request from another library, or educational institution, on SFU's own behalf or for one of its users. Distribution may be in any form.

The author has further agreed that SFU may keep more than one copy of the Work for purposes of back-up and security; and that SFU may, without changing the content, translate, if technically possible, the Work to any medium or format for the purpose of preserving the Work and facilitating the exercise of SFU’s rights under this licence.

It is understood that copying, publication, or public performance of the Work for commercial purposes shall not be allowed without the author's written permission.

While granting the above uses to SFU, the author retains copyright ownership and moral rights in the Work, and may deal with the copyright in the Work in any way consistent with the terms of this licence, including the right to change the Work for subsequent purposes, including editing and publishing the Work in whole or in part, and licensing the content to other parties as the author may desire.

The author represents and warrants that he/she has the right to grant the rights contained in this licence and that the Work does not, to the best of the author’s knowledge, infringe upon anyone’s copyright. The author has obtained written copyright permission, where required, for the use of any third-party copyrighted material contained in the Work. The author represents and warrants that the Work is his/her own original work and that he/she has not previously assigned or relinquished the rights conferred in this licence.

Simon Fraser University Library
Burnaby, British Columbia, Canada

revised Fall 2013
Abstract

As space-filling designs, orthogonal arrays have been widely used either directly or via OA-based Latin hypercubes in computer experiments. He and Tang (2013) introduced and constructed a new class of arrays, strong orthogonal arrays, for computer experiments. Strong orthogonal arrays of strength t enjoy better space-filling properties than comparable orthogonal arrays of strength t in all dimensions lower than t. Given a single orthogonal array, many strong orthogonal arrays can be generated using the method of He and Tang (2013). We examine the selection of better strong orthogonal arrays using the maximin distance, which is a criterion attempting to place points in a design region so that no two points are too close. In this project, we focus on maximin strong orthogonal arrays of strength three. For small designs, we apply the method of complete search. For large designs, the complete search is infeasible and we propose a search algorithm, which greatly reduces the computation time compared with the complete search approach. The performance of the algorithm is examined and it is found that the algorithm performs almost as well as the complete search.

Key words: Strong Orthogonal Array; Maximin Distance; Search Algorithm
To my family.
“And medicine, law, business, engineering, these are noble pursuits and necessary to sustain life.

But poetry, beauty, romance, love, these are what we stay alive for”

— John Keating, Dead Poets Society, 1989
Acknowledgments

First of all, I would like to express my greatest appreciation to my senior supervisor Dr. Boxin Tang for his patient guidance, kind encouragement and great mentorship. Without his continuous support and insightful suggestions, this work would not have been possible. His thorough understanding and attitudes towards research have always been inspiring me academically and mentally. It has been a real privilege to be his student.

I am also deeply grateful to the examining committe, Dr. Jiguo Cao, Dr. Michelle Zhou, and Dr. Tim Swartz. Thanks for taking your valuable time to supervise me in this project and offering helpful comments. I would like to thank Dr. Charmaine Dean for great mentorship and suggestions. She has inspired me greatly since my undergraduate study. I would like to thank all the other professors and instructors that taught me, helped me get ready for this project and understand more aspects of statistics, including Dr. Joan Hu, Dr. Dave Campbell Dr. Richard Lockhart and many others.

Big thanks to my friends at SFU for sharing ideas, wonderful memories and experience together: Eric Joel, Fei Wang, Yi Xiong, Kunasekaran Nirmalkanna, Biljana Stojkova, Elena Szefer, Huijing Wang, Annie Yu, Sabrina Zhang, Pulindu Ratnasekera, Wijendra Premarathna, Abdollah Safari, Luyao Lin, Jack Davis and many others. I would like to thank the Department of Statistics and Actuarial Science for offering such a nice graduate program and providing great learning and research atmosphere. I thank Sadika, Kelly and Charlene for help with the administrative matters.

Last but not least, I would like to thank my family, especially my parents and Eric for giving me endless love and support.
1 Introduction
1.1 Computer Experiments .................................................... 2
1.2 Space-Filling Designs ....................................................... 2
1.3 Latin Hypercube Designs .................................................. 3
1.4 Orthogonal Array-Based Latin Hypercubes ................................. 4
1.5 Strong Orthogonal Array-Based Latin Hypercubes ......................... 5
2 Distance-Based Design Selection Criteria

2.1 Space-Filling Criteria

2.2 Distance-Based Criteria

2.3 Minimax Distance Criterion

2.4 Maximin Distance Criterion and $\Phi_p$ Criterion

3 Maximin Strong Orthogonal Arrays by Complete Search

3.1 Method of Complete Search and an Illustrative Example

3.2 Search Results and Discussion

4 Maximin Strong Orthogonal Arrays by Algorithmic Search

4.1 Basic Idea

4.2 An Illustrative Example

4.3 Algorithm and Search Results

4.4 Performance of Algorithm

5 Concluding Remarks

Bibliography

Appendix A A Collection of Maximin Strong Orthogonal Arrays

A.1 Maximin SOAs in 8 Runs - Maximin $SOA(8, 3, 2^3, 3)$

A.2 Maximin SOAs in 16 Runs - Maximin $SOA(16, 7, 2^3, 3)$

A.3 Maximin SOAs in 27 Runs - Maximin $SOA(27, 3, 3^3, 3)$

Appendix B Maximin Strong Orthogonal Arrays from Algorithmic Search

B.1 Maximin SOAs in 54 Runs - Maximin $SOA(54, 4, 3^3, 3)$
## List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Table Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Complete search results for maximin SOAs</td>
<td>15</td>
</tr>
<tr>
<td>4.1</td>
<td>$\Phi$ values for the starting design and its neighbours</td>
<td>18</td>
</tr>
<tr>
<td>4.2</td>
<td>$\Phi$ values for the current design and its one-unit away neighbours</td>
<td>19</td>
</tr>
<tr>
<td>4.3</td>
<td>$\Phi$ values for the current design and its two-unit away neighbours</td>
<td>20</td>
</tr>
<tr>
<td>4.4</td>
<td>$\Phi$ values for the current design and its one-unit away neighbours</td>
<td>20</td>
</tr>
<tr>
<td>4.5</td>
<td>$\Phi$ values for the current design and its two-unit away neighbours</td>
<td>20</td>
</tr>
<tr>
<td>4.6</td>
<td>Distribution of $\Phi_p$ values for 27-run algorithmic search</td>
<td>23</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 2D projection of an SOA (also a Latin hypercube) ................................. 7
1.2 2D projection of an OA-based Latin hypercube ................................. 7

4.1 Illustrative pictures of designs and their neighbours ............................... 19
4.2 Histograms of $\Phi$ values for 16-run search ................................... 24
4.3 Histograms of $\Phi$ values for 27-run search ................................... 25
Chapter 1

Introduction

In scientific and engineering research, it is common that systems of interest are often very complicated. Conducting real experiments on such complicated systems can be too expensive or simply infeasible. These systems can be generally approximated by computer models, on which we can run computer experiments. Various models can be built using different methods to emulate the system of interest. We are interested in the relationship between the input and the output of this system. Many input variables can be manipulated over plausible ranges. Constrained by time and resources, we can only run very limited experimental trials. Before running a computer experiment, a key step is to determine what subset of the full input space should be considered, that is, how to select values of input variables for the experiments.

In this chapter, we discuss a general design approach to planning computer experiments and then review relevant concepts such as space-filling designs, Latin hypercubes, OA-based Latin hypercubes and SOA-based Latin hypercubes. A variety of optimality criteria are available to evaluate the space-filling properties of designs. We review the distance-based criteria and in particular the maximin distance criterion in Chapter 2. There are many strong orthogonal arrays available for the experimenter to use but they are not equally space-filling. We use the maximin distance criterion to select preferable strong orthogonal arrays and present the results based on a complete search
in Chapter 3. A complete search for maximin strong orthogonal arrays is feasible for small exam-
pies of strong orthogonal arrays. However, when the number of factors and the run size increase,
conducting a complete search becomes impractical due to limited computational capabilities under
time constraints, hence the need for developing a computationally efficient algorithm. In Chapter 4,
we introduce the algorithm and evaluate its performance via comparing with the complete search.
In Chapter 5, we conclude this project, provide suggestions for similar studies and also mention
possible future work.

1.1 Computer Experiments

A computer experiment is used to study a computer model and is conducted by running com-
puter codes that take a set of inputs and produce outputs. The output is usually referred to as
the response. Computer models can be either stochastic or deterministic. We limit our discus-
sion to deterministic models. For deterministic computer models, the three basic design principles,
randomization, replication and blocking, do not apply. Given the same inputs, the computer code
yields identical responses. In computer experiments, the uncertainty comes from the fact that we
do not know the exact relationship between the inputs and the response. Computer models built to
describe this relationship are just approximations. Two principles for selecting designs are:

1. There should not be more than one observation at any set of inputs. This principle assumes
   the computer code remains unchanged over time.

2. Designs should allow one to fit various models and should provide information about all por-
tions of the experimental region.

1.2 Space-Filling Designs

Designing a computer experiment is to select values of input variables before running the exper-
iment. Due to the lack of knowledge of the relationship between inputs and outputs, there exist
differences between the approximate computer models and the mathematical function representing
the physical experiments. To detect such differences, designs should provide information about all
portions of the experimental region. So the approach is to spread the design points throughout the
experimental region as evenly as possible. Such designs are referred to as space-filling designs.
There are a number of ways to define what it means by “spread points evenly” throughout a region,
which corresponds to different criteria for assessing space-filling designs. More will be discussed in
Chapter 2.

1.3 Latin Hypercube Designs

One most commonly used class of space-filling designs is that of Latin hypercubes. Latin hyper-
cube designs were introduced in McKay, Beckman and Conover (1979), and achieve uniformity in
all one-dimensional projections. In practice, an experiment usually involves a large number of input
variables. Due to the curse of dimensionality, it is rather difficult for design points to cover a large
portion of a high dimensional design region. Thus, it is more reasonable to consider designs that
are space-filling in lower dimensional projections of the input space. We now review the definition
of Latin hypercubes.

Definition 1.1. A Latin hypercube is a design for \( m \) factors in \( n \) runs and is represented by an
\( n \times m \) matrix, where each column is a permutation of \( n \) equally spaced levels \( x = 0, 1, 2, ..., n - 1 \).

Example 1.2. Two random Latin hypercubes \( X_1 \) and \( X_2 \), for three factors in five runs are given
below:

\[
X_1 = \begin{pmatrix}
1 & 1 & 2 \\
2 & 2 & 3 \\
3 & 4 & 1 \\
4 & 0 & 4 \\
0 & 3 & 0 \\
\end{pmatrix}, \quad X_2 = \begin{pmatrix}
3 & 2 & 4 \\
2 & 4 & 1 \\
4 & 0 & 3 \\
0 & 1 & 2 \\
1 & 3 & 0 \\
\end{pmatrix}
\]
1.4 Orthogonal Array-Based Latin Hypercubes

Orthogonal arrays were originally developed by Rao (1947). Orthogonal array-based Latin hypercube designs, also known as U designs, were proposed by Tang (1993). We will review the concept of orthogonal arrays and the construction of OA-based Latin hypercubes.

**Definition 1.3.** An $n \times m$ matrix is an $n$-run orthogonal array of strength $t$ and index $\lambda$ with $m$ factors each at $s$ levels if all possible $t$-tuples appear in each $n \times t$ submatrix with the same frequency $\lambda$ satisfying $n = \lambda s^t$. This orthogonal array is denoted by $OA(n, m, s, t)$.

A Latin hypercube is an orthogonal array of strength 1. An orthogonal array with mixed levels can be similarly defined, and is denoted by $OA(n, m, s_1 \times ... \times s_m, t)$.

**Definition 1.4.** Let $A$ be an $OA(n, m, s, t)$. For each column of $A$ with entry $k = 0, 1, ..., s - 1$, replace the $\lambda s^{t-1}$ entries for level $k$ with a random permutation of $k\lambda s^{t-1}$, $k\lambda s^{t-1} + 1$, ..., $k\lambda s^{t-1} + \lambda s^{t-1} - 1 = (k + 1)\lambda s^{t-1} - 1$. After the replacement, the design is an OA-based Latin hypercube.

**Example 1.5.** A strength two orthogonal array $A$ is given below, which is an $OA(4, 2, 2, 2)$. Based on this orthogonal array $A$, many Latin hypercubes can be generated. Designs $D_1$, $D_2$, $D_3$, $D_4$ and $D_5$ are just some randomly generated Latin hypercubes based on the orthogonal array $A$.

$$A = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}, \quad D_1 = \begin{pmatrix} 0 & 1 \\ 1 & 3 \\ 2 & 0 \\ 3 & 2 \end{pmatrix}, \quad D_2 = \begin{pmatrix} 0 & 0 \\ 1 & 3 \\ 3 & 1 \\ 2 & 2 \end{pmatrix}, \quad D_3 = \begin{pmatrix} 1 & 0 \\ 0 & 2 \\ 3 & 1 \\ 2 & 3 \end{pmatrix}, \quad D_4 = \begin{pmatrix} 1 & 1 \\ 0 & 3 \\ 2 & 0 \\ 3 & 2 \end{pmatrix}, \quad D_5 = \begin{pmatrix} 1 & 0 \\ 0 & 3 \\ 2 & 1 \end{pmatrix}.$$

Latin hypercubes achieve uniformity in all one-dimensional projections. A random Latin hypercube design is not guaranteed to achieve uniformity in multi-dimensional projections, but an OA-based Latin hypercube enjoys a multi-dimensional space-filling property. OA-based Latin hypercubes achieve uniformity in $t$-dimensional projections when orthogonal arrays of strength $t$ are used.
1.5 Strong Orthogonal Array-Based Latin Hypercubes

Definition 1.6. An \( n \times m \) matrix with entries from levels \( \{0, 1, ..., s^t - 1\} \) is a strong orthogonal array of \( n \) runs, \( m \) factors, \( s^t \) levels and strength \( t \) if any subarray of \( g \) columns for any \( g \) with \( 1 \leq g \leq t \) can be collapsed into an \( OA(n, g, s^{u_1} \times s^{u_2} \times ... \times s^{u_g}, g) \) for any positive integers \( u_1, ..., u_g \) with \( u_1 + ... + u_g = t \), where collapsing into \( s^{u_j} \) levels is done by dividing the entries by \( s^{t-u_j} \) and then taking the largest integer not exceeding the resulting entries, equivalently \( [a/s^{t-u_j}] \). We denote such a strong orthogonal array as \( SOA(n, m, s^t, t) \).

Example 1.7. A strength three strong orthogonal array \( D \) is given below, which is an \( SOA(8, 3, 2^3, 3) \). Collapsing the first two columns into \( 2 \times 4 \) and \( 4 \times 2 \) levels gives \( C_1 \) and \( C_2 \). Similarly, \( C_3 \) and \( C_4 \) are obtained from the first and the last columns. We see that \( C_1, C_3 \) are \( OA(8, 2, 2^2 \times 2, 2) \) and \( C_2, C_4 \) are \( OA(8, 2, 2^2 \times 2, 2) \).

\[
D = \begin{pmatrix}
2 & 0 & 0 \\
0 & 3 & 6 \\
1 & 6 & 2 \\
3 & 5 & 4 \\
4 & 2 & 3 \\
6 & 1 & 5 \\
7 & 4 & 1 \\
5 & 7 & 7 \\
\end{pmatrix}, \quad C_1 = \begin{pmatrix}
0 & 0 \\
0 & 1 \\
0 & 3 \\
0 & 2 \\
1 & 1 \\
1 & 2 \\
1 & 3 \\
1 & 3 \\
\end{pmatrix}, \quad C_2 = \begin{pmatrix}
1 & 0 \\
0 & 1 \\
0 & 0 \\
1 & 1 \\
2 & 0 \\
3 & 0 \\
3 & 1 \\
2 & 1 \\
\end{pmatrix}, \quad C_3 = \begin{pmatrix}
0 & 0 \\
0 & 3 \\
0 & 1 \\
0 & 2 \\
0 & 1 \\
1 & 1 \\
1 & 2 \\
1 & 3 \\
\end{pmatrix}, \quad C_4 = \begin{pmatrix}
1 & 0 \\
0 & 1 \\
0 & 0 \\
1 & 1 \\
1 & 2 \\
3 & 1 \\
3 & 0 \\
2 & 1 \\
\end{pmatrix}.
\]

Definition 1.8. Recall that a Latin hypercube of \( n \) runs for \( m \) factors is an \( n \times m \) matrix where each column is a permutation of \( 0, 1, ..., n - 1 \). The index of the strong orthogonal array \( \lambda \) is defined the same way as that of an orthogonal array. Since the index satisfies \( n = \lambda s^t \), an \( SOA(n, m, s^t, t) \) itself is a Latin hypercube when \( \lambda = 1 \). When \( \lambda \geq 2 \), the SOA is not a Latin hypercube but can be converted into one by expanding the \( s^t \) levels into \( n = \lambda s^t \) levels. For each column of the SOA, this is done by replacing the \( \lambda \) entries for level \( j \) by any permutation of \( j \lambda, j \lambda + 1, ..., (j + 1) \lambda - 1 \) for \( j = 0, 1, ..., s^t - 1 \). The Latin hypercubes generated this way is called \( SOA \)-based Latin hypercubes.
Strong orthogonal arrays (SOAs) were introduced and constructed by He and Tang (2013). An SOA\((n, m, s^1, t)\) promises a stratification of design points on an \(s^{u_1} \times \ldots \times s^{u_g}\) grid for any positive \(u_j\) with \(u_1 + \ldots + u_g = t\) on any \(g\)-variate margins, where \(g \leq t\). In addition to retaining all the projection space-filling properties of an SOA, a Latin hypercube based on an SOA\((n, m, s^1, t)\) maximizes the stratification for one-dimensional projections.

Both a Latin hypercube based on an SOA\((n, m, s^2, 2)\) and one based on an OA\((n, m, s, 2)\) achieve a stratification on an \(s \times s\) grid in any two-dimensional projection. Hence, Latin hypercubes based on the OA\((n, m, s, 2)\) are as space-filling in two-dimensional projections as Latin hypercubes based on the SOA\((n, m, s^2, 2)\). For strength three or higher, SOA-based Latin hypercubes are more space-filling than OA-based Latin hypercubes when projected on \(g\) dimensions, where \(1 < g < t\).

For two Latin hypercubes of strength three, one is based on an OA\((n, m, s^3, 3)\) and the other based on an SOA\((n, m, s^3, 3)\). Both achieve a stratification on an \(s \times s \times s\) grid on any trivariate margin. But the Latin hypercube based on the SOA\((n, m, s^3, 3)\) achieves stratifications on \(s^2 \times s\) and \(s \times s^2\) grids on any bivariate margin, while that based on the OA\((n, m, s, 3)\) ensures a stratification on an \(s \times s\) grid on any bivariate margin.

Figure 1.1 shows the two-dimensional projections of an SOA\((8, 3, 2^3, 3)\), design \(D\) from Example 1.7, which is also a Latin hypercube since \(\lambda = 1\). For first two columns, stratifications are achieved on \(2 \times 4\) grids as in (a) and \(4 \times 2\) grids as in (b); such stratifications also hold for any other two columns, 3 vs.1 and 3 vs. 2. Figure 1.2 provides the two-dimensional projections for a Latin hypercube based on an OA\((8, 3, 2, 3)\). This OA-based Latin hypercube achieves a stratification on \(2 \times 2\) grid as in (c). But stratifications on finer grids are not promised as in (b). Stratification on \(2 \times 4\) grids is achieved, but this is not guaranteed for all OA-based Latin hypercubes.
CHAPTER 1. INTRODUCTION

(a) SOA: 2 × 4 grid

(b) SOA: 4 × 2 grid

Figure 1.1: 2D projection of an SOA (also a Latin hypercube)

(a) OALH: 2 × 4 grid

(b) OALH: 4 × 2 grid

(c) OALH: 2 × 2 grid

(d) The OA-based Latin hypercube

Figure 1.2: 2D projection of an OA-based Latin hypercube
Chapter 2

Distance-Based Design Selection

Criteria

There is a large number of strong orthogonal arrays available and we suspect that they are not equally space-filling. To choose strong orthogonal arrays that have better space-filling properties, we need to adopt some optimality criteria. In Section 2.1, we give a general introduction to these criteria. In Sections 2.2, 2.3 and 2.4, we review distance-based criteria and discuss the minimax and maximin criteria. We will use the maximin criterion to select preferable arrays among a family of strong orthogonal arrays.

2.1 Space-Filling Criteria

One idea of finding more space-filling designs is to apply some optimality criteria to the available designs and select those with desirable space-filling properties. To evaluate the space-filling property of a design, several criteria have been developed in the literature. Some select points in the experimental region using certain sampling methods, others are based on a measure of distance between points, and still more look at how uniformly distributed the points are throughout a region. For a detailed discussion, see Santner, Williams and Notz (2003).
CHAPTER 2. DISTANCE-BASED DESIGN SELECTION CRITERIA

2.2 Distance-Based Criteria

We consider the criteria for selecting a design based on a measure that quantifies how “spread out” a set of points is. One way in which design points may be considered “spread out” is to avoid two design points that are “too” close together; another way in which designs points may be regarded as “spread out” is to make every point in the design space close to some point in the design. These are the fundamental ideas of maximin and minimax distance criteria, which were originally introduced in Johnson, Moore and Ylvisaker (1990).

More precisely, consider a design consisting of \( n \) distinct design points \( \{ \vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n \} \) in a \( d \)-dimensional space. The location of design points in the experimental region represents the site of chosen inputs in the plausible space for input variables. An important distance measure is the \( p \)th-order distance between two points \( \vec{x} \) and \( \vec{y} \) in the design region. We use the Euclidean and rectangular distances between two points, which is \( d_p \) for \( p = 2 \) and \( p = 1 \) respectively:

\[
d_p(\vec{x}, \vec{y}) = \left[ \sum_{i=1}^{d} |x_i - y_i|^p \right]^{1/p}, \quad p \geq 1.
\]

2.3 Minimax Distance Criterion

Suppose the firefighting department will be setting up new fire stations in a new region. The following considerations must be made when choosing the sites. Given that there is a fire, it should be obvious that a fire truck will be dispatched from the closest fire station to the scene of the fire. However, not all locations will be equidistant from its closest fire station, leaving those furthest from the nearest station to be in the worst case scenario should a fire occur there. Taking this into consideration, the department must now consider optimizing (minimizing) the longest (maximum) distance to travel from the closest (minimum) fire station for every location in the region so that the firefighting services are more evenly distributed throughout the region. This is in essence what the minimax distance criterion is aiming for.
The minimax criterion considers the distance from a given point in the design region to the closest design point. This is just the distance from a place to its closest fire station. We regard this distance as the distance between a point $\vec{x}$ and a design $D$:

$$d_p(\vec{x}, D) = \min_{\vec{x}_i \in D} d_p(\vec{x}, \vec{x}_i).$$

For any design $D$, the worst place will have a maximum distance to its closest fire station. Different designs will have different maximum distances to its closest fire stations, $\max_{\vec{x}} d_p(\vec{x}, D)$. A design is a minimax distance design if the maximum distance between arbitrary points $\vec{x}$ in the experimental region and the design is a minimum among all designs, i.e., $D^*$ is a minimax distance design if

$$\max_{\vec{x}} d_p(\vec{x}, D^*) = \min_{\text{all } D} \max_{\vec{x}} d_p(\vec{x}, D).$$

### 2.4 Maximin Distance Criterion and $\Phi_p$ Criterion

A maximin distance design is a design that maximizes the minimum inter-site distance $d_p(\vec{x}_i, \vec{x}_j)$ for $1 \leq i, j \leq n$ and $i \neq j$, equivalently

$$\max_{\text{all } D} \min_{\vec{x}_i, \vec{x}_j \in D} d_p(\vec{x}_i, \vec{x}_j).$$

It is undesirable to have design points that are too close, so the maximin criterion tries to avoid such cases. Maximin designs guarantee that no two points in the design are “too close”, which makes design points spread out over the experimental region.

Morris and Mitchell (1995) further discussed the maximin distance criterion for computer experiments and introduced a more elaborate version of the criterion. For a given design, consider the list of distinct distances between design points $(d_1, d_2, \ldots, d_q)$ sorted in ascending order. Let $J_j$ denote the number of pairs of design points separated by distance $d_j$. The index list $(J_1, J_2, \ldots, J_q)$ can be obtained. A design is maximin if
• among available designs, it maximizes $d_1$;
• among those, it also minimizes $J_1$;
• among those, it maximizes $d_2$;
• among those, it minimizes $J_2$;
... 
• among those, it maximizes $d_q$;
• among those it minimizes $J_q$.

Going through this selection process, we can select preferable designs according to the maximin criterion. We consider not only maximizing the shortest distance but also minimizing the number of pairs reaching this distance. This criterion further maximizes subsequent shorter distances and minimizes the number of pairs reaching these distances. This more elaborate version further distinguishes designs when they have the same minimum distance and the same number of pairs reaching the minimum distance.

However, this procedure is not efficient, so a computationally convenient surrogate is used instead. A scalar-value criterion function can be used to rank competing designs,

$$
\Phi_p(D) = \left[ \sum_{j=1}^{q} (J_j d_j^{-p}) \right]^{1/p} = \left( \sum_{j=1}^{m} d_j^{-p} \right)^{1/p}.
$$

where $p$ is a positive integer. Here $q$ represents the number of distinct distances between two design points while $m$ is the total number of point pairs, and clearly $m = \binom{n}{2}$. For large enough $p$, each term in the sum dominates subsequent terms; in other words, term $d_1^{-p}$ dominates that of $d_2^{-p}$; also $d_2^{-p}$ term dominates that of $d_3^{-p}$; ... ; and $d_{q-1}^{-p}$ term dominates that of $d_q^{-p}$. For any given sets of designs, the designs that minimize the $\Phi_p$ value are preferable designs. With a very large $p$, the $\Phi_p$ criterion is equivalent to the maximin distance criterion.
Chapter 3

Maximin Strong Orthogonal Arrays

by Complete Search

In this chapter, we search for preferable arrays among a class of strong orthogonal arrays (SOAs). The focus is on the case of strength three. The reason is as follows: strength two SOAs are no more space-filling than strength two OAs; SOAs of strength four or higher are too expensive to use in practice because two or more columns have to be sacrificed to obtain SOAs from an OA. Constructing strength three SOAs only sacrifices one column in an OA, which is a reasonable price to pay for obtaining more space-filling SOA-based Latin hypercubes.

3.1 Method of Complete Search and an Illustrative Example

Based on a given orthogonal array, we can produce a family of strong orthogonal arrays as proposed in He and Tang (2013). For a given ordinary orthogonal array $OA(n, m, s, 3)$, we denote the $i$th column of the array as $a_i$. Let $A = OA(n, m, s, 3) = (a_1, a_2, ..., a_m)$. Given $A$, an $n \times 3m'$ array $B$ can be constructed with $b_{ij}$'s obtained by (3.1)-(3.3). Here $m' = m - 1$.

$$B = \{(b_{11} b_{12} b_{13}); \ldots; (b_{m'1} b_{m'2} b_{m'3})\}$$
(b_{11}, ..., b_{m/1}) = (a_1, ..., a_{m'}) \quad (3.1)
(b_{12}, ..., b_{m/2}) = (a_{m}, ..., a_m) \quad (3.2)
(b_{13}, ..., b_{m/3}) = (a_2, ..., a_{m'}, a_1) \quad (3.3)

Then a strong orthogonal array \( D = (d_1, ..., d_{m'}) = \text{SOA}(n, m', s^3, 3) \) can be generated by

\[
d_i = \sum_{j=1}^{3} b_{ij} s^{3-j}.
\] (3.4)

We give an example to illustrate how to construct SOAs based on an OA. Consider \( A = \text{OA}(8, 4, 2, 3) = (a_1, a_2, a_3, a_4) \), we obtain an array \( B = \{(a_1 a_4 a_2); (a_2 a_4 a_3); (a_3 a_4 a_1)\} \). Then we generate \( D = \text{SOA}(8, 3, 2^3, 3) = (d_1, d_2, d_3) \) by

\[
\begin{align*}
d_1 &= 4a_1 + 2a_4 + a_2, \\
d_2 &= 4a_2 + 2a_4 + a_3, \\
d_3 &= 4a_3 + 2a_4 + a_1.
\end{align*}
\] (3.5)-(3.7)

In this example, various permutations of levels (symbols) within \( a_i \)'s in (3.5)-(3.7) can generate different corresponding SOAs. For example, based on \( A \) given below, we generate two SOAs \( D_1 \) and \( D_2 \). The first array is obtained by using (3.5)-(3.7). The only difference for the second is by permuting the symbols 0,1 in \( a_1 \) in (3.5).

\[
\begin{align*}
A &= \begin{pmatrix}
0 0 0 0 \\
0 0 1 1 \\
0 1 0 1 \\
0 1 1 0 \\
1 0 0 1 \\
1 0 1 0 \\
1 1 0 0 \\
1 1 1 1
\end{pmatrix},
D_1 &= \begin{pmatrix}
0 0 0 \\
2 3 6 \\
3 6 2 \\
1 5 4 \\
6 2 3 \\
4 1 5 \\
5 4 1 \\
7 7 7
\end{pmatrix},
D_2 &= \begin{pmatrix}
4 0 0 \\
6 3 6 \\
7 6 2 \\
5 5 4 \\
2 2 3 \\
0 1 5 \\
1 4 1 \\
3 7 7
\end{pmatrix}
\end{align*}
\]
After generating a family of SOAs, we apply the maximin distance criterion to select preferable SOAs. We call such preferable arrays maximin SOAs. The basic idea of complete search for maximin SOAs is to rank the set of SOAs using the $\Phi_p$ criterion introduced in Section 2.4. The designs with minimal $\Phi_p$ are optimal designs. We can tune the value of $p$ depending on the size of designs to ensure this criterion selects the maximin designs.

The complete search method has the following steps:

**Step I** Given an $OA(n, m, s, 3)$ with symbols 0, 1, ..., $s - 1$, generate an $SOA(n, m', s^3, 3)$ using (3.4) with $b_{ij}$'s defined as (3.1) - (3.3).

**Step II** Generate different SOAs by permuting symbols for the columns of $A$, $a_i$'s, within the formula for $d_j$. There are $s!$ permutations of symbols for each column of $A$ and there are $3m'$ positions of $a_i$ where we can permute symbols. So in total, $(s!)^{3m'}$ SOAs can be generated.

**Step III a** List $\binom{n}{2}$ distances between point pairs. Given a $p$ value, compute $\Phi_p$ for each SOA.

**Step IV a** Find the SOAs with the minimal $\Phi_p$ value and suppose there are $q$ such SOAs. These resulting $q$ SOAs are the maximin SOAs for an appropriately chosen $p$.

### 3.2 Search Results and Discussion

Using the complete search method, we have obtained the maximin SOAs of strength three for 8 runs, 16 runs and 27 runs. Based on the $\Phi_p$ value, it is difficult to understand how diverse the designs are in terms of their distance properties. To have a better understanding of how the generated SOAs vary, we provide the results using the variants b of Steps III and IV instead.

**Step III b** Compute the minimal distance and record the number of pairs reaching this distance.

**Step IV b** Find SOAs with the maximized minimal distance, and among these candidate SOAs, find those with the smallest number of point pairs reaching this distance.
Using the method above, we present a summary of the complete search results. For each size, the distance between point pairs is either the Euclidean distance or the rectangular distance. Among all SOAs generated, only a small portion are maximin SOAs: 32 out of 512 for 8 runs; 128 out of 262144 for 16 runs; 3 out of 46656 using the Euclidean distance and 2 out of 46656 using the rectangular distance for 27 runs. The generated SOAs seem to differ by much when comparing their minimal distances between point pairs. The minimal Euclidean distances between point pairs range from 1.73 to 4.12 for 8 runs, 2.65 to 7.14 for 16 runs, 1.73 to 8.77 for 27 runs; the minimal rectangular distances between point pairs range from 3 to 7 for 8 runs, 7 to 16 for 16 runs, 3 to 14 for 27 runs. If you generate an SOA randomly, it is most likely not maximin. From the complete search, the percentage of the SOAs that are maximin is around 6.25% for 8 runs, 0.05% for 16 runs and 0.006% for 27 runs.
Some of the maximin SOAs found may be geometrically isomorphic (Cheng and Ye, 2004). Two designs are geometrically isomorphic if one can be obtained from the other by reordering the runs, relabelling the factors and/or reversing the level order of one or more factors. For 8 runs, all the 32 maximin SOAs are found to be geometrically isomorphic; For 27 runs, there are in total two non-isomorphic maximin SOAs; For 16 runs, it is too time-consuming to check for geometrical isomorphism among the 128 maximin SOAs.

A collection of obtained maximin SOAs is provided in Appendix A, giving the experimenter the flexibility to apply a secondary criterion to select even more preferable arrays. For larger size SOAs, this complete search algorithm still works but is rather time consuming. To find maximin SOA of run size \(n\), \(m\) constraints each with \(s\) levels, \(SOA(n, m, s^3, 3)\), \((s!)^3m\) SOAs can be generated by permuting the symbols in each column of \(A\) in the formula. Complete search for maximin \(SOA(64, 3, 4^3, 3)\) requires calculating 2016 distances for each design, evaluating the \(\Phi_p\) values for 2,641,807,540,224 designs. The amount of evaluations increases exponentially every time \(m\) increases by 1. As the number of factors and the run size increase, the complete search can take an insurmountable amount of time. This justifies a need for a much more efficient method to search for larger maximin SOAs. So we will propose a computationally efficient algorithm to search for maximin SOAs in the next chapter.
Chapter 4

Maximin Strong Orthogonal Arrays
by Algorithmic Search

In the previous chapter, we have found maximin SOA designs of strength three for 3 factors in 8 runs, 7 factors in 16 runs and 3 factors in 27 runs. When the run size \( n > 27 \) or the number of factors \( m > 7 \), the complete search method requires a huge amount of computation. The proposed algorithm in this chapter takes much shorter time to obtain the optimal designs under the maximin criterion.

4.1 Basic Idea

Suppose a group of scientists wants to find the point with the lowest elevation in an area. They start with measuring the elevations of a chosen location \( S_0 \) and nine other locations \( S_1, S_2, ..., S_9 \) evenly spaced in the neighbourhood of \( S_0 \) with the same distance (1km) away from \( S_0 \). If the current location \( S_0 \) does not have the lowest elevation among the nine neighbouring locations, then the scientists will move to the new location that is the lowest among the nine. If there happens to be more than one neighbour having the lowest elevation, then just randomly choose one of these neighbours to move to. The idea is that as one moves towards lower and lower elevations, it becomes more and more likely to reach the lowest point in the area. The search will continue until the current location
is the lowest among its neighbours. Given that there is no new lowest location from the nine 1km away neighbours, the search will be expanded to the 2km away neighbourhood. If a new lowest location is detected, the scientists will move to this new location and search in a neighbourhood starting from 1km radius. Again, the scientists will keep moving to the new lowest location until no neighbour with lower elevation is found within 2km-radius neighbourhood. This final location is the lowest point found when starting from location \( S_0 \).

A useful strategy to increase the chance of finding the lowest point in an area is to start the search at different locations. Navigation starting from different points may lead to discovering other better local optimal points or the global optimal point. The example here limits the local search within a radius of 2km; one may expand the maximum search radius to 4km or even 6km. The radius is restricted because as you increase the maximum search radius, more candidate neighbours will be scanned and it can become increasingly computationally expensive. So it will be wise to flexibly modify this maximum search radius depending on the size of the radius relative to the whole area.

### 4.2 An Illustrative Example

The idea of the algorithm is fairly similar to the method of exploring the lowest point in an area. Consider the problem of searching for maximin SOAs of strength three in 8 runs, \( SOA(8, 3, 2^3, 3) \). Following Step I in Chapter 3, we start with a randomly generated SOA, \( D_0 \), obtained from an OA. The one-unit away neighbours around \( D_0 \) can be obtained by permuting symbols in just one column of the OA, \( a_i \), in the formulas (3.5)-(3.7). Compute the distances between any two points of \( D_0 \) and also those for the nine neighbours, which gives a sequence of \( \binom{8}{2} = 28 \) distances \( d_1, d_2, \ldots, d_{28} \) for each design. Then compute the \( \Phi_p \) value for each design. In this illustrative example, we use rectangular distance and \( p = 4 \) in Table 4.1.

<table>
<thead>
<tr>
<th>( \Phi_4(D) )</th>
<th>( D_0 )</th>
<th>( D_1 )</th>
<th>( D_2 )</th>
<th>( D_3 )</th>
<th>( D_4 )</th>
<th>( D_5 )</th>
<th>( D_6 )</th>
<th>( D_7 )</th>
<th>( D_8 )</th>
<th>( D_9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0.3193 )</td>
<td>0.3196</td>
<td>0.3332</td>
<td><strong>0.2993</strong></td>
<td>0.3196</td>
<td>0.3332</td>
<td><strong>0.2993</strong></td>
<td>0.3196</td>
<td>0.3332</td>
<td><strong>0.2993</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: \( \Phi \) values for the starting design and its neighbours
We see that the neighbours $D_3$, $D_6$ and $D_9$ have the smallest $\Phi_4$ value. Here we just move to one of these neighbours. Suppose that $D_3$ is randomly selected among the three best neighbours. See Figure 4.1 (a) for the illustrative picture: the large center circle (green) represents the initial design $D_0$ and surrounding smaller circle (blue) represent the one-unit away neighbours of $D_0$, with $D_3$ minimizing the $\Phi_4$ value. We then treat $D_3$ as our current optimal design, new $D_0$, denoted by $D_0'$, and obtain its nine new neighbours. Again, we compute the corresponding $\Phi_4$ values for this current design and its neighbours, as seen in Table 4.2. Comparing $D_0'$ with its one-unit away neighbours, it is still our current optimal design, which minimizes the $\Phi_4$ value. We have found the optimal design among its one-unit away neighbours, $D_0'$. But we cannot claim it as the optimal design yet.

To avoid searching within "too" small a neighbourhood for the lowest point in an area, we need to look into further away neighbourhood. Similarly, we need to check neighbours that are two units away from $D_0'$. So we expand the search from one-unit away neighbourhood (medium-size blue circle) to two-unit away neighbourhood (small red circle) as in Figure 4.1 (b). In other words, obtain $36$, i.e. $\binom{3m}{2}$ when $m = 3$, neighbouring designs and calculate their $\Phi_4$ values. From Table 4.3,
D_{11} and D_{24} minimize the \( \Phi_4 \) value among its two-unit away neighbours. Suppose \( D_{11} \) is randomly selected among the two best neighbours. Since \( D_{11} \) is treated as the current optimal design, we replace \( D_0' \) by \( D_{11} \). Then we move to \( D_0' \). For the current design \( D_0' \), we checked all of its one-unit and two-unit away neighbours and found that \( D_0' \) is the optimal design compared to one-unit and two-unit away neighbours (Tables 4.4 and 4.5) and we denote it as \( D_L \).

### 4.3 Algorithm and Search Results

The algorithm for finding maximin SOA\((n, m, s^3, 3)\) can be summarized as follows:

\[
\begin{array}{cccccccccccc}
\Phi_4(D) & D_0' & D_1 & D_2 & D_3 & D_4 & D_5 & D_6 & D_7 & D_8 & D_9 \\
0.2748 & 0.2993 & 0.3193 & 0.3196 & 0.3332 & 0.2993 & 0.3196 & 0.3657 & 0.3193 & 0.2748 \\
\Phi_4(D) & D_{10} & D_{11} & D_{12} & D_{13} & D_{14} & D_{15} & D_{16} & D_{17} & D_{18} & D_{19} \\
0.2993 & 0.3196 & 0.3332 & 0.2993 & 0.3196 & 0.3332 & 0.3657 & 0.3193 & 0.2993 & 0.2748 \\
\Phi_4(D) & D_{20} & D_{21} & D_{22} & D_{23} & D_{24} & D_{25} & D_{26} & D_{27} & D_{28} & D_{29} \\
0.3110 & 0.3196 & 0.3196 & 0.3110 & 0.3196 & 0.3196 & 0.3657 & 0.3657 & 0.2993 & 0.2748 \\
\Phi_4(D) & D_{30} & D_{31} & D_{32} & D_{33} & D_{34} & D_{35} & D_{36} & D_{37} & D_{38} & D_{39} \\
0.3110 & 0.3193 & 0.3110 & 0.3196 & 0.3196 & 0.3110 & 0.3193 & 0.3193 & 0.3193 & 0.3193 \\
\end{array}
\]

Table 4.5: \( \Phi \) values for the current design and its two-unit away neighbours
CHAPTER 4. MAXIMIN STRONG ORTHOGONAL ARRAYS BY ALGORITHMIC SEARCH

STEP A For given \( n, m, s \), obtain an OA\((n, m + 1, s, 3)\) using symbols 0, 1, ..., \( s - 1 \). For the \( (m + 1) \) columns, keep all the possible symbol permutations of each column in a list. Each column has \( s! \) symbol permutations. If \( s = 2 \), only two permutations are available, which is 01 and 10. If \( s = 3 \), there are six permutations available, that is 012, 021, 102, 120, 201, and 210.

STEP B Construct an initial SOA using (3.1) - (3.4) with a random choice of symbol permutation for the \( 3m \) \( a_i \)'s. Denote this initial SOA as \( D_0 \).

STEP C Obtain a list of the current SOA and its one-unit away neighbours. The length of this list is \( (3m + 1) \). The one-unit away neighbours are obtained by permuting symbols in one column of OA, i.e. \( a_i \) in (3.1) - (3.4). After obtaining the \( 3m \) such neighbours, compute the \( \Phi_p \) values for the designs in this list using the \( \Phi \) criterion function for a specified \( p \) value as given below.

\[
\Phi_p(D) = \left( \sum_{j=1}^{m} d_j^{-p} \right)^{1/p}.
\]

If there is more than one design that minimizes the \( \Phi_p \) value and the current one is not one of them, then replace the current design \( D_0 \) by one of the designs minimizing \( \Phi_p \).

STEP D Repeat STEP C until the current SOA \( D_0 \) is one of the designs with the smallest \( \Phi_p \) value among the one-unit away neighbours. Denote the current design as \( D_0 \) when the repeat stops.

STEP E Obtain a list of the current SOA and its two-unit away neighbours, the length of this list is \( \binom{3m}{2} + 1 \). The two-unit away neighbours are obtained by permuting symbols in any two columns of OA, i.e. \( a_i \) and \( a_j \) in (3.1) - (3.4). After obtaining the \( \binom{3m}{2} \) such neighbours, compute the \( \Phi_p \) values for the designs in this list.

STEP F If the current design \( D_0 \) does not minimize the \( \Phi_p \) value among its two-unit away neighbours, set \( D_0 \) to one of the designs minimizing \( \Phi_p \) and start from STEP C again. Otherwise, the current design \( D_0 \) is the optimal design found by the algorithm.
The complete search approach requires comparing \((s!)^3m\) designs. In contrast, the algorithm developed in this chapter requires comparing \(D_0\) with \(3m\) designs for Step C and \(\binom{3m}{2}\) designs for Step E. The algorithmic search requires 2 to 7 iterations for 8-run SOAs, 3 to 12 iterations for 16-run SOAs and 3 to 16 iterations for 27-run SOAs, showing a great reduction in the amount of computation.

There is no guarantee that the proposed algorithm will produce a maximin SOA (global optimal design) since the algorithm is based on a local search. As shown in the next section, our algorithm actually performs quite well, at least for the cases we have examined.

### 4.4 Performance of Algorithm

In Chapter 3, we have found the maximin SOA designs for 8 run, 16 runs, and 27 runs using the complete search. We will compare these results with those found using our algorithm. The comparison is based on the \(\Phi_p\) values. For 8-run SOAs for three factors, there are 512 SOAs that can be generated using the method in Chapter 3. Using our algorithm we can always find a maximin SOA for 8 runs. For 16-run SOAs for 7 factors, there are 262144 SOA designs in total. For each 16-run design, there are 21 one-unit away neighbours and 210 two-unit away neighbours. For 27-run SOAs for 3 factors, there are 46656 SOA designs that can be generated.

For each run size, we use the histogram of the \(\Phi_p\) values of all the generated SOAs to assess the performance of our algorithm. The vertical lines are the first quartile (green), median (blue), and the third quartile (orange) from left to right. We run our algorithm 100 times for each \(p\) value and the large red and the small brown triangles mark the positions of the design found with the largest \(\Phi_p\) (the worst case) and the smallest \(\Phi_p\) (the best case) among the hundred trials. We present the histograms for \(p = 1, 3, 6, 9\) (16-run SOAs) in Figure 4.2, and \(p = 1, 3, 5, 7\) (27-run SOAs) in Figure 4.3. We have also investigated other \(p\) values. All these results show that although our algorithm may not find the maximin design all the time, it always generates designs very close to the maximin.
To get a closer look at the performance of the algorithm, for SOAs of 27 runs, we run our algorithm 100 times and present the resulting distribution of $\Phi_p$ and their positions among the 46656 designs in Table 4.6. For almost all $p$ values investigated, designs at the first position (minimizing the $\Phi_p$ value) are found 20 out of 100 times in the worst case and 50 out of 100 times in the best case. It is also seen that the algorithm outputs the top-ranked designs in all cases.
CHAPTER 4. MAXIMIN STRONG ORTHOGONAL ARRAYS BY ALGORITHMIC SEARCH

Figure 4.2: Histograms of $\Phi$ values for 16-run search
CHAPTER 4. MAXIMIN STRONG ORTHOGONAL ARRAYS BY ALGORITHMIC SEARCH

25

(a) Euclidean distance, p=1
(b) Rectangular distance, p=1
(c) Euclidean distance, p=3
(d) Rectangular distance, p=3
(e) Euclidean distance, p=5
(f) Rectangular distance, p=5
(g) Euclidean distance, p=7
(h) Rectangular distance, p=7

Figure 4.3: Histograms of $\Phi$ values for 27-run search
Chapter 5

Concluding Remarks

This project considers searching for maximin strong orthogonal arrays of strength three. When the number of factors is seven or less and the run size is twenty-seven or less, we have conducted a complete search. For designs of larger sizes, the complete search requires too large an amount of computation. We have proposed a search algorithm to reduce computation time. The performance of our search algorithm has been examined and we have found that the algorithm performs quite well when compared with the complete search. We have applied the algorithm to search for larger maximin SOAs. See the 54-run SOAs found in Appendix B.

To improve the algorithm, one can simply consider neighbourhoods of larger than two units. An even better algorithm can be developed by allowing the size of neighbourhoods to vary, giving the flexibility to travel to potentially larger neighbourhood and scan more diverse neighbours. This can be done by assigning a probability \( p_j \) to size \( j \), where \( j = 1, 2, \ldots, g \), \( p_1 > p_2 > \ldots > p_g \), and \( \sum_{j=1}^{g} p_j = 1 \) if neighbourhoods of size \( g \) or less are considered. Rather than considering one-unit away neighbours before expanding to two-unit away neighbours, we randomly select a size before each iteration using \( p_1, p_2, \ldots, p_g \). This search approach will be considered in future studies.

As the number \( m \) of factors and the number \( s \) of symbols increase, the number of neighbours increases: \( 3m \times (s! - 1) \) one-unit away neighbours, \( \binom{3m}{2} \times (s! - 1)^2 \) for size two neighbours. When
$m = 10$ and $s = 3$, the total number of two-unit away neighbours, 10875, may be too large to be all examined. One may choose a portion of the totality. For example, we may randomly select 100 two-unit away neighbours to examine. A threshold may be set to decide when to stop the search. For example, for a threshold of 10, the search will claim it has found the optimal design if no movement is made for 10 iterations.
Bibliography


Appendix A

A Collection of Maximin Strong Orthogonal Arrays

A.1 Maximin SOAs in 8 Runs - Maximin \(SOA(8, 3, 2^3, 3)\)

\[
\begin{bmatrix}
4 & 2 & 0 \\
6 & 1 & 6 \\
7 & 4 & 2 \\
5 & 7 & 4 \\
2 & 0 & 3 \\
0 & 3 & 5 \\
1 & 6 & 1 \\
3 & 5 & 7 \\
\end{bmatrix}, \quad
\begin{bmatrix}
3 & 2 & 0 \\
1 & 1 & 6 \\
0 & 4 & 2 \\
2 & 7 & 4 \\
5 & 0 & 3 \\
7 & 3 & 5 \\
6 & 6 & 1 \\
4 & 5 & 7 \\
\end{bmatrix}, \quad
\begin{bmatrix}
4 & 5 & 0 \\
6 & 6 & 6 \\
7 & 3 & 2 \\
5 & 0 & 4 \\
2 & 7 & 3 \\
3 & 5 & 0 \\
1 & 1 & 1 \\
3 & 2 & 7 \\
\end{bmatrix}, \quad
\begin{bmatrix}
3 & 5 & 0 \\
1 & 6 & 6 \\
0 & 3 & 2 \\
2 & 0 & 4 \\
6 & 1 & 1 \\
7 & 4 & 5 \\
4 & 2 & 7 \\
5 & 7 & 3 \\
\end{bmatrix}, \quad
\begin{bmatrix}
2 & 0 & 4 \\
0 & 3 & 2 \\
1 & 6 & 6 \\
3 & 5 & 0 \\
4 & 2 & 7 \\
1 & 6 & 6 \\
2 & 7 & 3 \\
5 & 0 & 3 \\
\end{bmatrix}, \quad
\begin{bmatrix}
5 & 0 & 4 \\
7 & 3 & 2 \\
6 & 6 & 6 \\
4 & 5 & 0 \\
3 & 2 & 7 \\
1 & 1 & 6 \\
6 & 6 & 1 \\
7 & 3 & 5 \\
\end{bmatrix}, \quad
\begin{bmatrix}
2 & 7 & 4 \\
0 & 4 & 2 \\
1 & 1 & 6 \\
3 & 2 & 0 \\
4 & 5 & 7 \\
1 & 1 & 1 \\
6 & 6 & 1 \\
7 & 3 & 5 \\
\end{bmatrix}
\]
\[
\begin{align*}
(5, 7, 4) &\rightarrow (0, 4, 2) &\rightarrow (7, 4, 2) &\rightarrow (0, 3, 2) &\rightarrow (7, 3, 2) &\rightarrow (6, 6, 6) &\rightarrow (1, 6, 6) \\
7, 4, 2 &\rightarrow 2, 7, 4 &\rightarrow 5, 7, 4 &\rightarrow 2, 0, 4 &\rightarrow 5, 0, 4 &\rightarrow 4, 5, 0 &\rightarrow 3, 5, 0 \\
6, 1, 6 &\rightarrow 3, 2, 0 &\rightarrow 4, 2, 0 &\rightarrow 3, 5, 0 &\rightarrow 4, 5, 0 &\rightarrow 5, 0, 4 &\rightarrow 2, 0, 4 \\
4, 2, 0 &\rightarrow 1, 1, 6 &\rightarrow 6, 1, 6 &\rightarrow 1, 6, 6 &\rightarrow 6, 6, 6 &\rightarrow 7, 3, 2 &\rightarrow 0, 3, 2 \\
3, 5, 7 &\rightarrow 6, 6, 1 &\rightarrow 1, 6, 1 &\rightarrow 6, 1, 1 &\rightarrow 1, 1, 1 &\rightarrow 0, 4, 5 &\rightarrow 7, 4, 5 \\
1, 6, 1 &\rightarrow 4, 5, 7 &\rightarrow 3, 5, 7 &\rightarrow 4, 2, 7 &\rightarrow 3, 2, 7 &\rightarrow 2, 7, 3 &\rightarrow 5, 7, 3 \\
0, 3, 5 &\rightarrow 5, 0, 3 &\rightarrow 2, 0, 3 &\rightarrow 5, 7, 3 &\rightarrow 2, 7, 3 &\rightarrow 3, 2, 7 &\rightarrow 4, 2, 7 \\
2, 0, 3 &\rightarrow 7, 3, 5 &\rightarrow 0, 3, 5 &\rightarrow 7, 4, 5 &\rightarrow 0, 4, 5 &\rightarrow 1, 1, 1 &\rightarrow 6, 1, 1 \\
\end{align*}
\]
### APPENDIX A. A COLLECTION OF MAXIMIN STRONG ORTHOGONAL ARRAYS

$$\begin{pmatrix} 4 & 2 & 7 \\ 6 & 1 & 1 \\ 7 & 4 & 5 \\ 5 & 7 & 3 \\ 2 & 0 & 4 \\ 0 & 3 & 2 \\ 1 & 6 & 6 \\ 3 & 5 & 0 \end{pmatrix}, \begin{pmatrix} 3 & 2 & 7 \\ 1 & 1 & 1 \\ 0 & 4 & 5 \\ 2 & 7 & 3 \\ 5 & 0 & 4 \\ 7 & 3 & 2 \\ 6 & 6 & 6 \\ 4 & 5 & 0 \end{pmatrix}, \begin{pmatrix} 4 & 5 & 7 \\ 6 & 6 & 1 \\ 7 & 3 & 5 \\ 5 & 0 & 3 \\ 2 & 7 & 4 \\ 0 & 4 & 2 \\ 1 & 1 & 6 \\ 3 & 2 & 0 \end{pmatrix}, \begin{pmatrix} 3 & 5 & 7 \\ 1 & 6 & 1 \\ 0 & 3 & 5 \\ 2 & 0 & 3 \\ 7 & 4 & 2 \\ 6 & 1 & 6 \\ 4 & 2 & 0 \end{pmatrix}$$

#### A.2 Maximin SOAs in 16 Runs - Maximin SOA(16, 7, 2^3, 3)

Due to the limited space, only a subset of 128 maximin SOAs found is given here.

$$\begin{pmatrix} 4 & 0 & 0 & 1 & 5 & 5 & 5 \\ 6 & 2 & 3 & 7 & 6 & 2 & 3 \\ 6 & 3 & 6 & 2 & 3 & 6 & 3 \\ 4 & 1 & 5 & 4 & 0 & 1 & 5 \\ 7 & 6 & 2 & 2 & 2 & 3 & 7 \\ 5 & 4 & 1 & 4 & 1 & 4 & 1 \\ 5 & 5 & 4 & 1 & 4 & 0 & 1 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 \end{pmatrix}, \begin{pmatrix} 4 & 0 & 0 & 5 & 5 & 5 & 5 \\ 6 & 2 & 3 & 3 & 6 & 2 & 3 \\ 6 & 3 & 6 & 6 & 3 & 6 & 3 \\ 4 & 1 & 5 & 0 & 0 & 1 & 5 \\ 7 & 6 & 2 & 6 & 2 & 3 & 7 \\ 5 & 4 & 1 & 0 & 1 & 4 & 1 \\ 5 & 5 & 4 & 5 & 4 & 0 & 1 \\ 7 & 7 & 7 & 3 & 7 & 7 & 7 \end{pmatrix}, \begin{pmatrix} 4 & 0 & 0 & 1 & 3 & 5 & 5 \\ 6 & 2 & 3 & 7 & 0 & 2 & 3 \\ 6 & 3 & 6 & 2 & 5 & 6 & 3 \\ 4 & 1 & 5 & 4 & 6 & 1 & 5 \\ 7 & 6 & 2 & 2 & 4 & 3 & 7 \\ 5 & 4 & 1 & 4 & 7 & 4 & 1 \\ 5 & 5 & 4 & 1 & 2 & 0 & 1 \\ 7 & 7 & 7 & 1 & 7 & 7 \end{pmatrix}$$
| 4 0 0 7 7 1 5 | 4 0 0 0 5 5 5 | 4 0 0 3 1 5 5 |
| 6 2 3 1 4 6 3 | 6 2 3 6 6 2 3 | 6 2 3 5 2 2 3 |
| 6 3 6 4 1 2 3 | 6 3 6 3 3 6 3 | 6 3 6 0 7 6 3 |
| 4 1 5 2 2 5 5 | 4 1 5 5 0 1 5 | 4 1 5 6 4 1 5 |
| 7 6 2 4 0 7 7 | 7 6 2 3 2 3 7 | 7 6 2 0 6 3 7 |
| 5 4 1 2 3 0 1 | 5 4 1 5 1 4 1 | 5 4 1 6 5 4 1 |
| 5 5 4 7 6 4 1 | 5 5 4 0 4 0 1 | 5 5 4 3 0 0 1 |
| 7 7 7 1 5 3 7 | 7 7 7 6 7 7 7 | 7 7 7 5 3 7 7 |
| 3 7 7 0 0 6 2 | 3 7 7 7 2 2 2 | 3 7 7 4 6 2 2 |
| 1 5 4 6 3 1 4 | 1 5 4 1 1 5 4 | 1 5 4 2 5 5 4 |
| 1 4 1 3 6 5 4 | 1 4 1 4 4 1 4 | 1 4 1 7 0 1 4 |
| 3 6 2 5 5 2 2 | 3 6 2 2 7 6 2 | 3 6 2 1 3 6 2 |
| 0 1 5 3 7 0 0 | 0 1 5 4 5 4 0 | 0 1 5 7 1 4 0 |
| 2 3 6 5 4 7 6 | 2 3 6 2 6 3 6 | 2 3 6 1 2 3 6 |
| 2 2 3 0 1 3 6 | 2 2 3 7 3 7 6 | 2 2 3 4 7 7 6 |
| 0 0 0 6 2 4 0 | 0 0 0 1 0 0 0 | 0 0 0 2 4 0 0 |

(4 0 0 1 4 5 5) (4 0 0 2 5 1 5) (4 0 0 0 5 5 5)
(6 2 3 7 7 2 3) (6 2 3 4 6 6 3) (6 2 3 6 1 3 2 3)
(6 3 6 2 2 6 3) (6 3 6 1 3 2 3) (4 1 5 7 0 5 5)
(4 1 5 4 1 1 5) (4 1 5 7 0 5 5) (7 6 2 2 3 3 7)
(7 6 2 2 3 3 7) (7 6 2 1 2 7 7) (5 4 1 4 0 4 1)
(5 4 1 4 0 4 1) (5 4 1 7 1 0 1) (5 5 4 1 5 0 1)
(5 5 4 1 5 0 1) (5 5 4 2 4 4 1) (7 7 7 6 7 7 7)
(7 7 7 6 7 7 7) (7 7 7 4 7 3 7) (3 7 7 6 3 2 2)
(3 7 7 6 3 2 2) (3 7 7 5 2 6 2) (1 5 4 0 0 5 4)
(1 5 4 0 0 5 4) (1 5 4 3 1 1 4) (1 4 1 5 5 1 4)
(1 4 1 5 5 1 4) (1 4 1 6 4 5 4) (3 6 2 3 6 6 2)
(3 6 2 3 6 6 2) (3 6 2 0 7 2 2) (0 1 5 5 4 4 0)
(0 1 5 5 4 4 0) (0 1 5 6 5 0 0) (2 3 6 3 7 3 6)
(2 3 6 3 7 3 6) (2 3 6 0 6 7 6) (2 2 3 6 2 7 6)
(2 2 3 6 2 7 6) (2 2 3 5 3 3 6) (0 0 0 0 1 0 0)
(0 0 0 0 1 0 0) (0 0 0 3 0 4 0) (0 0 0 6 2 4 0)
A.3 Maximin SOAs in 27 Runs - Maximin $SOA(27, 3^3, 3)$

Maximin SOAs: $D_1, D_2, D_3$ using Euclidean distance; $D_1$ and $D_3$ using rectangular.

\[
D_1 = \begin{pmatrix}
8 & 8 & 8 \\
2 & 1 & 11 \\
1 & 11 & 2 \\
11 & 2 & 1 \\
5 & 3 & 23 \\
3 & 23 & 5 \\
23 & 5 & 3 \\
4 & 13 & 14 \\
14 & 4 & 13 \\
13 & 14 & 4 \\
16 & 16 & 16 \\
0 & 18 & 20 \\
20 & 0 & 18 \\
18 & 20 & 0 \\
24 & 24 & 24 \\
15 & 26 & 7 \\
17 & 6 & 25 \\
7 & 15 & 26 \\
25 & 17 & 6 \\
26 & 7 & 15 \\
6 & 25 & 17 \\
10 & 9 & 19 \\
9 & 19 & 10 \\
19 & 10 & 9 \\
12 & 21 & 22 \\
22 & 12 & 21 \\
21 & 22 & 12
\end{pmatrix},
D_2 = \begin{pmatrix}
5 & 5 & 5 \\
8 & 7 & 17 \\
7 & 17 & 8 \\
17 & 8 & 7 \\
2 & 0 & 20 \\
0 & 20 & 2 \\
20 & 2 & 0 \\
1 & 10 & 11 \\
11 & 1 & 10 \\
10 & 11 & 1 \\
13 & 13 & 13 \\
6 & 24 & 26 \\
26 & 6 & 24 \\
24 & 26 & 6 \\
21 & 21 & 21 \\
12 & 23 & 4 \\
14 & 3 & 22 \\
4 & 12 & 23 \\
22 & 14 & 3 \\
23 & 4 & 12 \\
3 & 22 & 14 \\
16 & 15 & 25 \\
15 & 25 & 16 \\
25 & 16 & 15 \\
9 & 18 & 19 \\
19 & 9 & 18 \\
25 & 15 & 24 \\
18 & 19 & 9
\end{pmatrix},
D_3 = \begin{pmatrix}
2 & 2 & 2 \\
5 & 4 & 14 \\
4 & 14 & 5 \\
14 & 5 & 4 \\
8 & 6 & 26 \\
6 & 6 & 26 \\
26 & 6 & 6 \\
7 & 16 & 17 \\
17 & 7 & 16 \\
16 & 7 & 7 \\
10 & 10 & 10 \\
3 & 21 & 23 \\
23 & 3 & 21 \\
21 & 23 & 3 \\
18 & 18 & 18 \\
9 & 20 & 1 \\
11 & 0 & 19 \\
1 & 9 & 20 \\
19 & 11 & 0 \\
20 & 1 & 9 \\
0 & 19 & 11 \\
13 & 12 & 22 \\
12 & 22 & 13 \\
22 & 13 & 12 \\
15 & 24 & 25 \\
25 & 15 & 24 \\
24 & 25 & 15
\end{pmatrix}
\]
Appendix B

Maximin Strong Orthogonal Arrays
from Algorithmic Search
### B.1 Maximin SOAs in 54 Runs - Maximin $SOA(54, 4, 3^3, 3)$

The transpose of the designs are as follows:

<table>
<thead>
<tr>
<th>$D_1^T$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>15</td>
<td>7</td>
<td>6</td>
<td>25</td>
<td>3</td>
<td>22</td>
<td>23</td>
<td>4</td>
<td>5</td>
<td>13</td>
<td>2</td>
<td>0</td>
<td>19</td>
<td>10</td>
<td>18</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>18</td>
<td>18</td>
<td>2</td>
<td>12</td>
<td>14</td>
<td>14</td>
<td>3</td>
<td>3</td>
<td>22</td>
<td>15</td>
<td>15</td>
<td>17</td>
<td>16</td>
<td>17</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>10</td>
<td>6</td>
<td>17</td>
<td>25</td>
<td>17</td>
<td>25</td>
<td>16</td>
<td>20</td>
<td>9</td>
<td>1</td>
<td>9</td>
<td>1</td>
<td>10</td>
<td>3</td>
<td>23</td>
<td>14</td>
<td>12</td>
<td>22</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>10</td>
<td>22</td>
<td>4</td>
<td>13</td>
<td>5</td>
<td>14</td>
<td>12</td>
<td>1</td>
<td>19</td>
<td>10</td>
<td>18</td>
<td>9</td>
<td>11</td>
<td>25</td>
<td>7</td>
<td>7</td>
<td>25</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1</th>
<th>9</th>
<th>12</th>
<th>9</th>
<th>10</th>
<th>10</th>
<th>18</th>
<th>18</th>
<th>2</th>
<th>12</th>
<th>14</th>
<th>14</th>
<th>3</th>
<th>3</th>
<th>22</th>
<th>15</th>
<th>15</th>
<th>17</th>
<th>16</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>7</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>15</td>
<td>7</td>
<td>6</td>
<td>25</td>
<td>3</td>
<td>22</td>
<td>23</td>
<td>4</td>
<td>5</td>
<td>13</td>
<td>2</td>
<td>0</td>
<td>19</td>
<td>10</td>
<td>18</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>10</td>
<td>6</td>
<td>17</td>
<td>25</td>
<td>17</td>
<td>25</td>
<td>16</td>
<td>20</td>
<td>9</td>
<td>1</td>
<td>9</td>
<td>1</td>
<td>10</td>
<td>3</td>
<td>23</td>
<td>14</td>
<td>12</td>
<td>22</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>13</td>
<td>10</td>
<td>22</td>
<td>4</td>
<td>13</td>
<td>5</td>
<td>14</td>
<td>12</td>
<td>1</td>
<td>19</td>
<td>10</td>
<td>18</td>
<td>9</td>
<td>11</td>
<td>25</td>
<td>7</td>
<td>7</td>
<td>25</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1  | 6  | 24 | 6  | 24 | 8  | 25 | 11 | 11 | 0  | 0  | 1  | 20 | 1  | 20 | 13 | 13 | 21 | 21 | 23 | 4  |
| 2  | 1  | 1  | 0  | 2  | 19 | 10 | 26 | 24 | 8  | 6  | 16 | 25 | 17 | 26 | 12 | 14 | 3  | 5  | 22 | 13 |
| 3  | 14 | 12 | 22 | 4  | 13 | 13 | 8  | 24 | 8  | 24 | 15 | 15 | 7  | 7  | 18 | 2  | 18 | 2  | 11 | 11 |
| 4  | 6  | 26 | 15 | 17 | 17 | 17 | 4  | 22 | 3  | 21 | 21 | 23 | 12 | 14 | 19 | 1  | 20 | 2  | 2  | 0  |

<table>
<thead>
<tr>
<th>41</th>
<th>42</th>
<th>43</th>
<th>44</th>
<th>45</th>
<th>46</th>
<th>47</th>
<th>48</th>
<th>49</th>
<th>50</th>
<th>51</th>
<th>52</th>
<th>53</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>23</td>
<td>4</td>
<td>2</td>
<td>19</td>
<td>19</td>
<td>22</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>2</td>
<td>21</td>
<td>12</td>
<td>24</td>
<td>15</td>
<td>17</td>
<td>14</td>
<td>23</td>
<td>21</td>
<td>20</td>
<td>11</td>
<td>9</td>
<td>9</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>19</td>
<td>19</td>
<td>26</td>
<td>26</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>3</td>
<td>5</td>
<td>21</td>
<td>23</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>9</td>
<td>3</td>
<td>5</td>
<td>23</td>
<td>20</td>
<td>18</td>
<td>0</td>
<td>24</td>
<td>6</td>
<td>24</td>
<td>8</td>
<td>26</td>
</tr>
<tr>
<td>$D^L_2$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>1</td>
<td>19</td>
<td>25</td>
<td>19</td>
<td>18</td>
<td>18</td>
<td>10</td>
<td>10</td>
<td>2</td>
<td>25</td>
<td>26</td>
<td>26</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>17</td>
<td>9</td>
<td>20</td>
<td>19</td>
<td>11</td>
<td>10</td>
<td>2</td>
<td>16</td>
<td>8</td>
<td>6</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>20</td>
<td>1</td>
<td>9</td>
<td>1</td>
<td>9</td>
<td>0</td>
<td>13</td>
<td>5</td>
<td>21</td>
<td>5</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
<td>18</td>
<td>3</td>
<td>12</td>
<td>21</td>
<td>13</td>
<td>22</td>
<td>23</td>
<td>9</td>
<td>0</td>
<td>18</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>13</td>
<td>4</td>
<td>13</td>
<td>5</td>
<td>12</td>
<td>20</td>
<td>20</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>5</td>
<td>23</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>10</td>
<td>20</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>8</td>
<td>15</td>
<td>24</td>
<td>6</td>
<td>6</td>
<td>19</td>
<td>11</td>
<td>19</td>
<td>11</td>
<td>2</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>7</td>
<td>26</td>
<td>25</td>
<td>26</td>
<td>25</td>
<td>12</td>
<td>3</td>
<td>14</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>23</td>
</tr>
</tbody>
</table>

`41 42 43 44 45 46 47 48 49 50 51 52 53 54`

| 1     | 17  | 6   | 2   | 9   | 9   | 15  | 8   | 8   | 5   | 3   | 3   | 12  | 14  | 14  |
| 2     | 7   | 25  | 1   | 19  | 18  | 24  | 6   | 7   | 3   | 21  | 22  | 22  | 4   | 3   |
| 3     | 12  | 12  | 10  | 10  | 20  | 23  | 23  | 13  | 26  | 25  | 17  | 16  | 17  | 25  |
| 4     | 19  | 20  | 14  | 13  | 4   | 1   | 2   | 11  | 8   | 17  | 8   | 16  | 7   | 16  |