Optimal Designs – Theory, Algorithms and Applications

by

Hongzhi Wang

(Under the Direction of Abhyuday Mandal)

Abstract

Design of Experiments plays an important role in modern science and engineering applications. It is one of the core fields in statistics area, and it is also a powerful tool in investigating new or existing processes to gain further insights. Optimal designs of experiments, which optimize the performances of different processes or target on maximizing the information gained from the limited data, are widely desired and used. Depending on the physical constraints and experimental requirements, various types of optimal designs are implemented, including D-optimal designs, space-filling designs, orthogonal designs and order-of-addition designs. It is often very challenging to construct optimal designs with flexible sizes, and theoretical results only exist for certain design sizes.

This dissertation focuses on optimal designs of experiments. More specifically, optimal space-filling designs, orthogonal designs, and order-of-addition designs in experiments with discrete elements, as well as D-optimal and G-optimal designs in experiments with continuous elements. For each type of the optimal designs, corresponding theories, related algorithms, applications, and my contributions to this area will be discussed.

Index words: Design of Experiments, Algorithms, Computer Experiments, Swarm intelligence, G-optimality, D-optimality

Optimal Designs – Theory, Algorithms and Applications

by

Hongzhi Wang

M.S., The University of Toledo, 2015
B.B.A., The University of Toledo, 2013

A Dissertation Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree.

Doctor of Philosophy

Athens, Georgia

2022

Optimal Designs – Theory, Algorithms and Applications

by

Hongzhi Wang

Major Professor:	Abhyuday Mandal
Committee:	Qian Xiao
	Gauri Datta
	Franklin West

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
August 2022

Acknowledgments

I would like to express my gratitude to Dr. Abhyuday Mandal, Dr. Qian Xiao, Dr. Gauri Datta and Dr. Franklin West. Thank you all for being my advisory committee members, thank you all for your incessant guidance and help, and thank you all for being part of my life.

I would also like to express my deepest appreciation to my wife, Yingying Liu, for her perpetual support, encouragement, patience, and forbearance at all times. She encouraged me to keep going when I felt like giving up and she supported me in many aspects to take my pressure off. I am so lucky to have her in my life.

Contents

Acknowledgments **iv**

List of Figures **vi**

List of Tables **vii**

1 Exploration on a Newly-published Algorithm, the Beginning of My Research **1**
1.1 Introduction . . . 1
1.2 Gray Wolf Optimizer . . . 1
1.3 Illustrative Example: Minimizing a Benchmark Function . . . 4
1.4 Research Outcomes . . . 5

2 Musings and an All-encompassing R library for Optimal Latin Hypercube Designs **6**
2.1 Introduction . . . 6
2.2 Optimality Criteria for LHDs . . . 8
2.3 Search Algorithms for Optimal LHDs with Flexible Sizes . . . 10
2.4 Algebraic Constructions for Optimal LHDs with Certain Sizes . . . 19
2.5 Numerical Results and Comparisons . . . 26
2.6 Conclusion and Recommendation . . . 36
2.7 Research Outcomes . . . 38

3 Lioness Algorithm for Finding Optimal Design of Experiments **39**
3.1 Introduction . . . 39
3.2 Brief Review on Optimal Designs . . . 42
3.3 The Lioness Algorithm for Optimal Designs . . . 45
3.4 Performance of the Proposed Algorithm . . . 52
3.5 Discussion . . . 62
3.6 Research Outcomes . . . 64

4 Identification of Predictive MRI and Functional Biomarkers in a Pediatric Piglet Traumatic Brain Injury Model **65**
4.1 Introduction 65
4.2 Materials and Methods 66
4.3 Results 70
4.4 Discussion 72
4.5 Conclusion 77
4.6 Research Outcomes 77

5 Conclusion **78**

Bibliography **79**

List of Figures

2.1 Boxplots of ϕ_p Values from Different Algorithms with the Rule of Thumb Sizes. 29
2.2 Convergence Curves for Different Algorithms with the Rule of Thumb Sizes. 30
2.3 Boxplots of ψ values from Different Algorithms with the Rule of Thumb Design Sizes. 32

3.1 Lionesses Hunting Positions 46
3.2 Boxplots of ϕ_p Values for LA and SLHD with the Rule of Thumb Sizes. 53
3.3 Convergence Curves for LA and SLHD with the Rule of Thumb Sizes. 54
3.4 Boxplots of ϕ_p Values for LA and SLHD with the Rule of Thumb Size. 55
3.5 Boxplots of ψ Values for LA and MaxProLHD with the Rule of Thumb Sizes. 57
3.6 Boxplots of D-optimality Criterion for LA with Different α Values. 63

4.1 MRI Images and PCA Plots. 71
4.2 Gait Images and PCA Plots. 73
4.3 Linear Regression Plots. 74

List of Tables

2.1 Summary Table of Simulated Annealing Based Algorithms. 11
2.2 Search algorithm functions in the LHD package 16
2.3 Summary Table of Run and Factor Sizes for Different Construction Methods. 21
2.4 Algebraic construction methods in the LHD package 24
2.5 Minimum ϕ_p and Average CPU Time (in seconds) under the L_1 Distance. 26
2.6 Minimum ϕ_p and Average CPU Time (in seconds) with Run Size Between 4 and 13 under the L_2 Distances. 27
2.7 Minimum ϕ_p and Average CPU Time (in minutes) with the Rule of Thumb Sizes under the L_2 Distances. 28
2.8 Minimum ψ and Average CPU Time (in seconds) with Run Size Between 4 and 14. . . 31
2.9 Minimum ψ and Average CPU Time (in minutes) with the Rule of Thumb Sizes. . . . 31
2.10 Minimum ave($|q|$), max$|q|$, and Average CPU Time (in seconds) with Run Size Below 2^4. 34
2.11 Minimum ave($|q|$), max$|q|$, and Average CPU Time (in seconds) with Run Size Between 2^4 and $2^5 - 1$. 34
2.12 Minimum ave($|q|$), max$|q|$, and Average CPU Time (in minutes) with Run Size Between 2^5 and 2^8. 35
2.13 Minimum ave($|q|$), max$|q|$, and Average CPU Time (in minutes) with the Rule of Thumb Sizes. 35
2.14 Recommended Algorithm (or Method) for Maximin Distance LHDs. 36
2.15 Recommended Algorithm for Maximum Projection LHDs. 37
2.16 Recommended Algorithm (or Method) for Orthogonal and Nearly Orthogonal LHDs 37

3.1 Average CPU Time (in seconds) for the Simulation in Figure 3.2. 52
3.2 Average CPU Time (in seconds) for the Simulation in Figure 3.5. 56
3.3 Minimum max$|q|$ and Average CPU time (in seconds) under Different Design Sizes. . . 56
3.4 D-efficiencies and Average CPU time (in seconds) of LA with different sizes for OofAs . 59
3.5 Relative G-efficiency for Different Algorithms with Respect to GA (Borkowski, 2003). 61

4.1 Gait Parameters Definition. 69
4.2 Linear Model Results. 72

Chapter 1

Exploration on a Newly-published Algorithm, the Beginning of My Research

1.1 Introduction

In recent decades, meta-heuristic optimization techniques became very popular in different scientific fields. Famous optimization techniques such as genetic algorithm (J. Holland, 1975), particle swarm optimization (Kennedy & Eberhart, 1995) and ant colony optimization (Dorigo et al., 2006) etc. are widely used. Gray Wolf Optimizer (Mirjalili et al., 2014) introduced in this chapter is a relatively new algorithm. The authors tested 29 different benchmark functions with different dimensions, and they found that GWO has better performances than other current popular algorithms. Motivated by the powerfulness of GWO, my advisor and I started exploring this algorithm and we wanted to see if it can be applied for practical applications with promising results.

1.2 Gray Wolf Optimizer

1.2.1 Optimization Problem Set-up

Suppose we are interested in finding the minimum of an objective function $f : \mathcal{R}^n \to \mathcal{R}$ where the j-th variable of interest x_j has the domain $\mathcal{D}_j = [L_j, U_j]$ for $j = 1, \ldots, n$, where L_j and U_j denote the lower bound and upper bound of x_j, respectively. A typical candidate solution will be given by $\mathbf{x} = (x_1, \ldots, x_n)^T \in \mathcal{X}$, where $\mathcal{X}$ is the collection of all possible candidates. Here $\mathcal{X} \in \mathcal{D}_1 \times \ldots \times \mathcal{D}_n$. Then the goal is to find the optimal solution $\mathbf{x}^*$ that minimizes the objective function f, such that

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmin}} f(\mathbf{x}). \tag{1.1}$$

We will use Gray Wolf Optimization technique to solve this type of problem and further identify optimal design of experiments.

1.2.2 Algorithm Overview and Mathematical Model

Gray wolf (scientific name *Canis lupus*), is a wild animal found in Eurasia and North America. It is a carnivorous animal and occupies the top position of the food chain. They are social animals with group size between 5 and 12 on average. The social dominant hierarchy among gray wolves is very strict. The hunting process of gray wolves can be divided into three phases, which are tracking and approaching the prey, encircling and harassing the prey until it stops, and attacking the prey.

Gray Wolf Optimizer (GWO) is a global optimization technique that can be used for identifying optimal designs, and it is inspired by the hunting process of gray wolves. There are several terminologies in GWO which are frequently used throughout this chapter. Search agents stands for the candidate solutions in the GWO algorithm. Position matrix in GWO algorithm is a collection of finite candidate solutions, which would be initialized at the first place and then updated over iterations. Score in GWO algorithm implies the objective function output of a certain candidate solution $\mathbf{x}$, namely $f(\mathbf{x})$.

Next is the demonstration of the social hierarchy of gray wolves and how it relates to the concept of finding optimal solution by GWO. The leader in a wolf group is called alpha (denoted by α), who makes decisions during the hunting. In the GWO, alpha is considered as the global best among all the search agents, which means α score is the global optimization $f(\mathbf{x}^*)$.

Beta (denoted by β) belongs to the second level in the hierarchy, who helps the alpha in making decisions. The beta wolf is the best candidate to replace alpha when alpha dies or becomes old. In the GWO, beta is the second best among all the search agents, which will be used to update the position matrix at each iteration.

The third level in the hierarchy is called delta (denoted by δ), who follows the orders from alpha and beta. In the GWO, delta is the third best among all the search agents, which will be used to update the position matrix together with the beta at each iteration. The lowest ranking in hierarchy is omega (denoted by ω), who plays the role of soldiers. Omega wolves follow the orders from all other dominant wolves. In nature, omegas are the last group that are allowed to eat, and sometimes they babysit newly born wolves. In the GWO, omega are the search agents except alpha, beta and delta.

The very first step of GWO is to initialize α, β, and δ scores by tentatively setting them equal to infinity for minimization problems or negative infinity for maximization problems. The second step is to initialize the position matrix, denoted by X_P. Suppose we have m search agents and n variables of interest, the dimension of X_P would be m by n, where he number of search agents, m, is an user-defined tuning parameter. Let $X_P^{(i,j)}$ denote the element from i^{th} row and j^{th} column of X_P, $i = 1, \ldots, m$, $j = 1, \ldots, n$, the $X_P^{(i,j)}$ is initialized according to the following formula,

$$r^{(i,j)} \times (U_j - L_j) + L_j \tag{1.2}$$

where $r^{(i,j)}$ are independent random draws from the standard uniform distribution (aka. Uniform$(0,1)$), and L_j and U_j are the lower bound and upper bound of x_j, respectively. The third step contains two parts.

1. **Find the Search Agents α, β, and δ.**

   Plugging in all the elements of each row of X_P into the objective function f, which yields m different function outputs, denoted by $f_i, i = 1, \ldots, m$. Suppose we are interested in finding the minimum of an objective function f, then the smallest f_i will be chosen as α score, the second smallest f_i will be chosen as β score, and the third smallest f_i will be chosen as δ score. At the same time, the corresponding row vectors for α score, β score, and δ score will be recorded as $\mathbf{x}_\alpha$, $\mathbf{x}_\beta$, and $\mathbf{x}_\delta$. The coding logic is given below. For $i = 1, \ldots, m$,

$$f_i = \begin{cases} \alpha\ score & \text{if } f_i < \text{previous } \alpha \text{ score,} \\ \beta\ score & \text{if } f_i > \text{previous } \alpha \text{ score but } f_i < \text{previous } \beta \text{ score,} \\ \delta\ score & \text{if } f_i > \text{previous } \beta \text{ score but } f_i < \text{previous } \delta \text{ score.} \end{cases}$$

2. **Update Position Matrix X_P.**

   For $i = 1, \ldots, m$ and $j = 1, \ldots, n$, the elements of position matrix X_P is updated by the following formula at each iteration:

$$X_P^{(i,j)} = \frac{\mathbf{x}_1^{(j)} + \mathbf{x}_2^{(j)} + \mathbf{x}_3^{(j)}}{3} \tag{1.3}$$

   where $\mathbf{x}_1^{(j)}$, $\mathbf{x}_2^{(j)}$, and $\mathbf{x}_3^{(j)}$ are defined as:

$$\begin{aligned} \mathbf{x}_1^{(j)} &= \mathbf{x}_\alpha^{(j)} - A_1^{(i,j)} \times |C_1^{(i,j)} \times \mathbf{x}_\alpha^{(j)} - X_P^{(i,j)}|, \\ \mathbf{x}_2^{(j)} &= \mathbf{x}_\beta^{(j)} - A_2^{(i,j)} \times |C_2^{(i,j)} \times \mathbf{x}_\beta^{(j)} - X_P^{(i,j)}|, \\ \mathbf{x}_3^{(j)} &= \mathbf{x}_\delta^{(j)} - A_3^{(i,j)} \times |C_3^{(i,j)} \times \mathbf{x}_\delta^{(j)} - X_P^{(i,j)}|, \end{aligned}$$

   where $A_1^{(i,j)}$, $A_2^{(i,j)}$, $A_3^{(i,j)}$, $C_1^{(i,j)}$, $C_2^{(i,j)}$, and $C_3^{(i,j)}$ are constant numbers such that,

$$\begin{aligned} a &= 2 \times (1 - \tfrac{\text{Current Iteration}}{\text{Total Number of Iteration}}), \\ A_1^{(i,j)} &= a \times (2 \times r1_1^{(i,j)} - 1), \\ A_2^{(i,j)} &= a \times (2 \times r1_2^{(i,j)} - 1), \\ A_3^{(i,j)} &= a \times (2 \times r1_3^{(i,j)} - 1), \\ C_1^{(i,j)} &= 2 \times r2_1^{(i,j)}, \\ C_2^{(i,j)} &= 2 \times r2_2^{(i,j)}, \\ C_3^{(i,j)} &= 2 \times r2_3^{(i,j)}, \end{aligned}$$

where $r1_1^{(i,j)}$, $r1_2^{(i,j)}$, $r1_3^{(i,j)}$, $r2_1^{(i,j)}$, $r2_2^{(i,j)}$, and $r2_3^{(i,j)}$ are independent random draws from the standard uniform distribution.

After all the elements of position matrix X_P have been updated, repeat the two parts above until maximum number of iterations have achieved. The α score after the algorithm stops would be the global optimization and the $\mathbf{x}_\alpha$ is the optimal solution $\mathbf{x}^*$.

1.3 Illustrative Example: Minimizing a Benchmark Function

Consider the following objective function:

$$f(\mathbf{x}) = \frac{20 + \sum x_i^2}{\sum (x_i - \bar{x})^2}.$$

The goal is to find the optimal solution that minimizes above equation under a 20-dimensional setting, and the domain for each x_i, $i = 1, \ldots, 20$, is $[0, 1]$. To implement GWO, the first step is to initialize α, β, and δ scores by tentatively setting them equal to infinity. The second step is to initialize the position matrix X_P. Suppose we determine to use 5 search agents, so the dimension of X_P would be 5 by 20. Let $X_P^{(i,j)}$ denote the element from i^{th} row and j^{th} column of X_P, $i = 1, \ldots, 5$, $j = 1, \ldots, 20$, the $X_P^{(i,j)}$ is initialized according to the following:

$$r^{(i,j)} \times (1 - 0) + 0 = r^{(i,j)},$$

where $r^{(i,j)}$ is an random draw from the standard uniform distribution. Suppose we set the number of iterations to be 100 and let $X_P^{[0]}$ denote the position matrix before the first iteration, $X_P^{[1]}$ denote the position matrix at the first iteration, . . ., $X_P^{[100]}$ denote the position matrix at the one hundred iteration. $X_P^{[0]}$ may look like the following,

$$X_P^{[0]} = \begin{bmatrix} 0.9187 & 0.1406 & 0.6968 & \ldots & 0.3241 \\ 0.5715 & 0.5984 & 0.5147 & \ldots & 0.9904 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.2468 & 0.8389 & 0.8397 & \ldots & 0.9423 \end{bmatrix}$$

For the first part of step three, which is to find the search agents α, β, and δ, we start with plugging in all the elements of each row of $X_P^{[0]}$ into the objective function. There should be 5 different function outputs, denoted by $f_i, i = 1, \ldots, 5$. The smallest f_i will be chosen as α score, the second smallest f_i will be chosen as β score, the third smallest f_i will be chosen as δ score, and their corresponding vectors will be recorded as $\mathbf{x}_\alpha$, $\mathbf{x}_\beta$, and $\mathbf{x}_\delta$. For example, the α, β, and δ scores as well as $\mathbf{x}_\alpha$, $\mathbf{x}_\beta$, and $\mathbf{x}_\delta$ may look like the following:

$\mathbf{x}_\alpha = (0.5715, 0.5984, 0.5147, 0.9000, \ldots, 0.8431, 0.3666, 0.0602, 0.9904)$ with α score 12.8113.

$\mathbf{x}_\beta = (0.3114, 0.1753, 0.1553, 0.7977, \ldots, 0.6894, 0.3997, 0.6574, 0.0326)$ with β score 14.8582.

$\mathbf{x}_\delta = (0.2468, 0.8389, 0.8397, 0.5627, \ldots, 0.6221, 0.0187, 0.5864, 0.9423)$ with δ score 18.4507.

For the second part of step three, which is updating the position matrix, we implement the corresponding equations, and the position matrix becomes $X_P^{[1]}$, which may look like the following:

$$X_P^{[1]} = \begin{bmatrix} 0.0000 & 0.7837 & 1.0000 & \ldots & 0.5597 \\ 0.2282 & 0.4883 & 0.3102 & \ldots & 0.7829 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.7595 & 0.8695 & 0.1762 & \ldots & 1.0000 \end{bmatrix}$$

Then we repeat the thrid step until the maximum number of iterations have achieved. $X_P^{[100]}$, $\mathbf{x}_\alpha$, and α score may end up being:

$$X_P^{[100]} = \begin{bmatrix} 0.0488 & 0.9965 & 0.0153 & \ldots & 0.9948 \\ 0.0491 & 1.0000 & 0.0153 & \ldots & 1.0000 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0492 & 1.0000 & 0.0154 & \ldots & 0.9976 \end{bmatrix}$$

$\mathbf{x}_\alpha = (0.0516, 1.0000, 0.0149, 1.0000, \ldots, 0.0370, 0.0385, 0.0370, 1.0000)$ with α score 6.5743. If we increase the number of search agents and the maximum of iterations, the result would be further improved, but note that the CPU time would increase accordingly.

1.4 Research Outcomes

After being familiar with the GWO algorithm, we implemented and tested it on different design problems. However, the results were not as accurate or improved as we expected. Despite the algorithm performance turns out not being promising, I would say the learning process is as important as scientific result. This was my first time reading academic papers, and this was also the first time I coded an algorithm in R and MatLab. Besides, I learned how to use Linux commands to submit computation scripts to university advanced computing resource center for parallel computing. Through this research, I learned the structure and background knowledge of several popular algorithms and I enhanced my coding skills in different statistical software environment. More importantly, I learned how to think critically from the perspective of design of experiments. The knowledge and experience I gained from this research built a good foundation for my future work.

Chapter 2

Musings and an All-encompassing R library for Optimal Latin Hypercube Designs

2.1 Introduction

Computer experiments are widely used in both scientific researches and industrial productions, where complex computer codes, commonly high-fidelity simulators, generate data in lieu of real physical systems (Fang et al., 2005; Sacks et al., 1989). Outputs from computer experiments are deterministic (i.e. free from random errors), and hence replications are not needed (Santner et al., 2003). Latin hypercube designs (LHDs, McKay et al., 1979) may be the most popular type of experimental designs for computer experiments (Fang et al., 2005; Xiao & Xu, 2018), which avoid replications on every dimension and have uniform one-dimensional projections. According to practical needs, there are various types of optimal LHDs, including space-filling LHDs, maximum projection LHDs and orthogonal LHDs. There is a rich literature on the construction of such designs, but it is still very challenging to find good ones for moderate to large design sizes (Fang et al., 2005; Joseph et al., 2015; Xiao & Xu, 2018; Ye, 1998).

An LHD with n runs and k factors is an $n \times k$ matrix with each column being a random permutation of numbers: $1, \ldots, n$. Throughout this chapter, n denotes the run size and k denotes the factor size. A space-filling LHD has its sampled region as scattered-out as possible, minimizing the unsampled region, thus accounting for the uniformity of all dimensions. Different criteria were proposed to measure designs' space-filling properties, including the maximin and minimax distance criteria (Johnson et al., 1990; Morris & Mitchell, 1995), the discrepancy criteria (Fang et al., 2005; Fang et al., 2002; Hickernell, 1998) and the entropy criterion (Shewry & Wynn, 1987). Since there are as many as $(n!)^{k-1}$ candidate LHDs for a given design size, it is nearly impossible to find the space-filling one via enumeration when n and k are moderate or large. In the current literature, both search algorithms (Ba et al., 2015; R.-B. Chen et al., 2013; Grosso et al., 2009; Jin et al., 2005; Joseph & Hung, 2008; Leary et al., 2003; Liefvendahl & Stocki, 2006; Morris

& Mitchell, 1995; Ye et al., 2000) and algebraic constructions (L. Wang et al., 2018; Xiao & Xu, 2017; Zhou & Xu, 2015) are used to construct space-filling LHDs.

Search algorithms are often used to identify space-filling LHDs with flexible sizes. Specifically, Morris and Mitchell, 1995 proposed a simulated annealing (SA) algorithm which can avoid being trapped at local optima and find the global best designs. Following their work as well as that of Tang, 1993, Leary et al., 2003 proposed to search for space-filling orthogonal array-based LHDs (OALHDs). Joseph and Hung, 2008 proposed a multi-objective criterion and developed an adapted SA algorithm, which considers both designs' orthogonality and space-filling properties. Ba et al., 2015 extended the work of sliced Latin hypercube designs (SLHD, Qian, 2012) and proposed a two-stage SA algorithm. In addition to these SA based algorithms, various types of search algorithms are proposed to find optimal designs. Ye et al., 2000 proposed the columnwise pairwise (CP) algorithm to identify efficient symmetric LHDs. Jin et al., 2005 proposed the enhanced stochastic evolutionary algorithm (ESE), which is a combination of an exchange procedure (inner loop) and a threshold determination method (outer loop). Liefvendahl and Stocki, 2006 proposed to use a genetic algorithm (GA), and implemented a strategy that directly works with the global best. Grosso et al., 2009 proposed to use an iterated local search heuristic where local searching and perturbation are considered. R.-B. Chen et al., 2013 proposed a version of particle swarm optimization (PSO) algorithm whose search process is to gradually reduce the Hamming distances between each particle and the global best (or personal best) via exchanging elements. Search algorithms are often computationally expensive, especially for constructing large designs. For some design sizes, algebraic constructions are available which may generate optimal LHDs with very low computational cost. For example, Xiao and Xu, 2017 adopted Costas' arrays to obtain space-filling saturate LHDs or Latin squares with run sizes $n = p - 2, p - 1$ or p, where p is any prime or prime power. L. Wang et al., 2018 proposed to apply the Williams transformation (E. Williams, 1949) to linearly permuted good lattice point sets to construct maximin LHDs when the factor sizes are no greater than the numbers of positive integers that are co-prime to the run sizes.

Space-filling designs often focus on the full dimensional spaces. To further improve the space-filling properties of all possible sub-spaces, Joseph et al., 2015 proposed to use the maximum projection designs. Considering from two to $k - 1$ dimensional sub-spaces, maximum projection LHDs (MaxPro LHDs) are generally more space-filling compared to the classic maximin distance LHDs. The construction of MaxPro LHDs is also challenging, especially for large ones, and Joseph et al., 2015 proposed an SA based algorithm. In this chapter, we propose a new GA framework that can lead to many better MaxPro LHDs; see Section 2.5 for examples.

Different from space-filling LHDs which minimize the similarities among rows, orthogonal LHDs (OLHDs) are another popular type of optimal designs which consider the similarities among columns. All search algorithms introduced above can be used to identify OLHDs which have zero column-wise correlations. Algebraic constructions are available for certain design sizes. For example, Ye, 1998 proposed an algebraic construction of OLHDs with run sizes $n = 2^m + 1$ and factor sizes $k = 2m - 2$ with m being any integer no less than 2. Cioppa and Lucas, 2007 further extended this approach to accommodate more factors. Steinberg and Lin, 2006 developed a construction method based on factorial designs and

group rotations with $n = 2^{2^m}$ and $k = 2^m t$, where m is any positive integer and t is the number of rotation groups. Sun et al., 2010 extended their earlier work (Sun et al., 2009) to construct OLHDs with $n = r2^{c+1}$ or $n = r2^{c+1} + 1$ and $k = 2^c$ where c and r are any two positive integers. J. Yang and Liu, 2012 proposed to use generalized orthogonal designs to construct OLHDs and nearly orthogonal LHDs (NOLHDs) with $n = 2^{r+1}$ or $n = 2^{r+1} + 1$ and $k = 2^r$ where r is any positive integer. Georgiou and Efthimiou, 2014 proposed a construction method based on orthogonal arrays and their full fold-overs with $n = 2ak$ runs and k factors, where k is the size of orthogonal matrix and a is any positive integer. Butler, 2001 implemented the Williams transformation (E. Williams, 1949) to construct OLHDs under a second-order cosine model with n being odd primes and $k \leq n - 1$. Additionally, inheriting the properties of orthogonal arrays, some new kinds of optimal LHDs can be constructed. For example, Tang, 1993 proposed to construct OALHDs by deterministically replacing elements from orthogonal arrays, which tend to have better space-filling proprieties with low column-wise correlations. C. D. Lin et al., 2009 proposed to use OLHDs or NOLHDs as starting designs and then couple them with orthogonal arrays for constructing OALHDs. Their method requires fewer runs to accommodate large numbers of factors with n^2 runs and $2fp$ factors, where n and p are designs sizes of the OLHDs or NOLHDs and $2f$ is the number of columns in the coupled orthogonal array.

We developed an integrated tool, the R package `LHD` available on the Comprehensive R Archive Network (https://cran.r-project.org/web/packages/LHD/index.html), which improves and efficiently implements some currently popular search algorithms and algebraic methods for constructing maximin distance LHDs, Maxpro LHDs, OLHDs and NOLHDs. We thoroughly review each method, describe our implementations and compare all discussed methods to provide guidance for practitioners. With this package, users with little or no background on design theory can easily find optimal LHDs with the required sizes. Many new designs that are better than the existing ones are discovered; see Section 2.5. Moreover, we review, summarize and compare some currently popular methods in the rich design literature aiming to provide guidance for practitioners.

The rest of the chapter is organized as follows. Section 2.2 illustrates different optimality criteria for LHDs. Section 2.3 demonstrates the details of some popular search algorithms and their implementations. Section 2.4 discusses some useful algebraic constructions. Section 2.5 provides numerical results and comparisons generated by the developed `LHD` package. Section 2.6 concludes with some discussions on the future research.

2.2 Optimality Criteria for LHDs

There are various criteria proposed to measure designs' space-filling properties (Fang et al., 2002; Hickernell, 1998; Johnson et al., 1990). In this chapter, we focus on the currently popular maximin distance criterion (Johnson et al., 1990), which seeks to scatter design points over the experimental domains such that the minimum distances between points are maximized. Let $\mathbf{X}$ denote an LHD matrix throughout this chapter. Define the L_q-distance between two runs x_i and x_j of $\mathbf{X}$ as $d_q(x_i, x_j) = \left\{\sum_{m=1}^{k} |x_{im} - x_{jm}|^q\right\}^{1/q}$ where q is an integer. Define the L_q-distance of design $\mathbf{X}$ as $d_q(\mathbf{X}) = \min\{d_q(x_i, x_j), 1 \leq i < j \leq n\}$.

In this chapter, we consider $q = 1$ and $q = 2$, i.e. the Manhattan (L_1) and Euclidean (L_2) distances. A design $\mathbf{X}$ is called a maximin L_q-distance design if it has the unique largest $d_q(\mathbf{X})$ value among all designs of the same size. When more than one designs have the same largest $d_q(\mathbf{X})$, the maximin distance design sequentially maximizes the next minimum inter-site distances. To evaluate the maximin distance criterion in a more convenient way, Morris and Mitchell, 1995 and Jin et al., 2005 proposed to minimize a scalar value:

$$\phi_p = \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_q(x_i, x_j)^{-p} \right\}^{1/p}, \tag{2.1}$$

where p is a tuning parameter. This ϕ_p criterion in Equation (2.1) is asymptotically equivalent to the Maximin distance criterion as $p \to \infty$. In practice, $p = 15$ often suffices. In the `LHD` package, function `phi_p()` implements this criterion.

Maximin distance LHDs focus on the space-filling properties in the full dimensional space, but their space-filling properties in the sub-spaces are not guaranteed. Joseph et al., 2015 proposed the maximum projection criterion that considers designs' space-filling properties in all possible dimensional spaces. An LHD $\mathbf{X}$ is called a maximum projection LHD (MaxPro LHD) if it minimizes the maximum projection criterion such that

$$\min_{\mathbf{X}} \psi(\mathbf{X}) = \left\{ \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{\Pi_{l=1}^{k} (x_{il} - x_{jl})^2} \right\}^{1/k}. \tag{2.2}$$

From Equation (2.2), we can see that any two design points should be apart from each other in any projection to minimize the value of $\psi(\mathbf{X})$. Thus, the maximum projection LHDs consider the space-filling properties in all possible sub-spaces. In the `LHD` package, function `MaxProCriterion()` implements this criterion.

Orthogonal and nearly orthogonal designs which aim to minimize the correlations between factors are widely used in experiments (Georgiou, 2009; Steinberg & Lin, 2006; Sun & Tang, 2017). Two major correlation-based criteria to measure designs' orthogonality is the average absolute correlation criterion and the maximum absolute correlation criterion (Georgiou, 2009), denoted as ave$(|q|)$ and max$|q|$, respectively:

$$\text{ave}(|q|) = \frac{2 \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} |q_{ij}|}{k(k-1)} \text{ and } \max |q| = \max_{i,j} |q_{ij}|, \tag{2.3}$$

where q_{ij} is the correlation between the ith and jth columns of the design matrix $\mathbf{X}$. Orthogonal designs have ave$(|q|) = 0$ and max$|q| = 0$, which may not exist for all design sizes. Designs with smaller ave$(|q|)$ or max$|q|$ are generally preferred in practice. In the `LHD` package, functions `AvgAbsCor()` and `MaxAbsCor()` implement the ave$(|q|)$ and max$|q|$ criteria, respectively.

2.3 Search Algorithms for Optimal LHDs with Flexible Sizes

2.3.1 Simulated Annealing Based Algorithms

Simulated annealing (SA, Kirkpatrick et al., 1983) is a probabilistic optimization algorithm, whose name comes from the phenomenon of annealing process in metallurgy. It simulates the process of heating a material beyond a critical temperature and then gradually cooling it in a controlled manner in order to minimize its defects. SA is a single-solution based algorithm that starts with one random solution and iteratively improve it until convergence. It is widely used for optimization problems with continuous factors. As the elements in LHDs are discrete, researchers modified the standard SA for identifying optimal LHDs.

Morris and Mitchell, 1995 proposed a modified SA which randomly exchanges the elements in LHDs to seek potential improvements. Specifically, it starts with a randomly generated LHD from which a random column is chosen. Then, two random elements in this column are exchanged to form a new LHD. For example, in an LHD with 4 runs, suppose that a randomly chosen column has the elements: $(1, 2, 3, 4)$. If we exchange its first two elements, this column becomes $(2, 1, 3, 4)$, which leads to a new LHD. If such an exchange leads to a better LHD under a given optimality criterion, we should keep this exchange. Otherwise, we should keep it with a probability of $\exp[-(\Phi(\mathbf{X}_{new}) - \Phi(\mathbf{X}))/T]$, where Φ is a given optimality criterion, $\mathbf{X}$ is the original LHD, $\mathbf{X}_{new}$ is the LHD after the exchange and T is the current temperature. Such an exchange procedure will be iteratively implemented to improve the LHD. When there are no improvements after certain attempts, the current temperature T would be annealed to increase the probability of updating the LHD. The best LHD is identified after the algorithm converges or the budget constraint is reached. The pseudo-code for our implementation of SA is outlined in Algorithm 1. In the `LHD` package, function `SA()` implements this algorithm.

Algorithm 1 Simulated Annealing Algorithm for LHDs

1: Generate one random LHD, denoted as $\mathbf{X}$.
2: **while** not converge **do**
3: Randomly choose a column from $\mathbf{X}$, denoted as j.
4: Exchange two randomly selected elements within column j, and denote the new LHD as $\mathbf{X}_{new}$.
5: If $\Phi(\mathbf{X}_{new}) < \Phi(\mathbf{X})$, then $\mathbf{X} = \mathbf{X}_{new}$. Otherwise, let $\mathbf{X} = \mathbf{X}_{new}$ with a probability of $\exp[-\frac{\Phi(\mathbf{X}_{new})-\Phi(\mathbf{X})}{T}]$.
6: When no improvements are found consecutively for certain attempts, decrease current temperature T and repeat Steps 3−5.
7: **end while**

Leary et al., 2003 modified the SA algorithm in Morris and Mitchell, 1995 to search optimal orthogonal array-based LHDs (OALHDs). Tang, 1993 showed that OALHDs tend to have better space-filling properties than random LHDs. The SA in Leary et al., 2003 starts with a random OALHD from which a random column is chosen. Two random elements that share the same entry in the original orthogonal array (OA) are exchanged. For example, in an OALHD with 9 runs, suppose that a randomly chosen column

Table 2.1: Summary Table of Simulated Annealing Based Algorithms.

	Morris95	Leary03	Joseph08
Starting Design	A random LHD	A random OALHD	A random LHD
Column Choice	A random column	A random column	A column who has the largest average pairwise correlation
Elements Choice	Two random elements	Two random elements share the same original OA entry	The element from the row having largest total row-wise distance with a random element in the same column

has the elements: $(1, 2, 3, 4, 5, 6, 7, 8, 9)$ and its original OA has the entries: $(\mathit{1}, \mathit{1}, \mathit{1}, \mathit{2}, \mathit{2}, \mathit{2}, \mathit{3}, \mathit{3}, \mathit{3})$. Elements $(1, 2, 3)$ share the same original OA entry $\mathit{1}$, $(4, 5, 6)$ share the same original OA entry $\mathit{2}$, and $(7, 8, 9)$ share the same original OA entry $\mathit{3}$. The remaining steps are the same as the SA in Morris and Mitchell, 1995. Note that the existence of OALHDs is determined by the existence of the corresponding initial OAs. In the `LHD` package, function `OASA()` implements the modified SA algorithm.

Joseph and Hung, 2008 proposed another modified SA to identify the orthogonal-maximin LHDs, which considers both the orthogonality and the maximin distance criteria. The algorithm starts with generating a random LHD and then chooses the column which has the largest average pairwise correlations with all other columns. As an illustration, the algorithm selects the l^*th column where $l^* = \text{argmax}\, \rho_l^2$ and $\rho_l^2 = \frac{1}{k-1} \sum_{j \neq l} \rho_{lj}^2$ is the average pairwise correlation between the lth and all other columns. Next, the algorithm will select the row which has the largest total row-wise distance with all other rows. As an illustration, the algorithm selects the i^*th row where $i^* = \text{argmax}(\sum_{j \neq i} d(x_i, x_j)^{-p})^{1/p}$ and $d(x_i, x_j)$ is the distance between the ith and jth rows. Then, the element at the i^*th row and j^*th column will be exchanged with a random element from the same column. The remaining steps are the same as the SA in Morris and Mitchell, 1995. This algorithm in Joseph and Hung, 2008 leads to optimal LHDs minimizing both ϕ_p and ave$(|q|)$ criteria, but it is computationally heavy, since it needs to calculate all ρ_l^2 values and row-wise distances at each iteration. In the `LHD` package, function `SA2008()` implements this algorithm. In Table 2.1, we summarize the similarities and differences of the three SA algorithms introduced above.

Ba et al., 2015 and Qian, 2012 proposed to construct space-filling sliced Latin hypercube designs (SLHDs). An n-run SLHD can be partitioned into t slices of sub-matrices ($\mathbf{X} = \bigcup_{i=1}^{t} \mathbf{X}_i$) where each slice $\mathbf{X}_i$ is also an LHD with $m = n/t$ runs. They propose an SA algorithm consisting of two stages. The first stage aims to optimize the design matrix from the slice perspective. It starts with t random slices i.e. $m \times k$ LHDs. If there are duplicated rows among the slices, the algorithm will randomly pick a duplicated row, then choose another random row in the same slice, and finally swap the two elements in these two rows and a randomly chosen column. Repeat this procedure until no duplicated rows exists. Next, the algorithm will exchange the elements from a random column in a random slice. If such an exchange leads

to duplicated rows, the previous step will be repeated; otherwise, keep this exchange when there is an improvement. The second stage optimizes the design matrix from the element level via replacing t of level l ($l = 1, \ldots, m$) with a random permutation of $\{(l-1)t+1, \ldots, lt\}$ and then exchanging the two randomly selected elements within the permutation. When $t = 1$, this algorithm targets on finding the classic space-filling LHDs. In the `LHD` package, function `SLHD()` implements this algorithm.

2.3.2 Particle Swarm Optimization Algorithms

Particle swarm optimization (PSO, Kennedy and Eberhart, 1995) is a meta-heuristic optimization algorithm inspired by social behaviors of animals, e.g. the flying pattern in a bird flock. PSO is a population-based algorithm that solves problems by having a population of particles (candidate solutions), moving them around the search space according to some mathematical formulae over the particle's position and velocity, and iteratively trying to improve the candidate solution in terms of a given optimality measure. The movement of each particle is affected by its local best known position, and it is also guided moving towards the best positions in the search space which are updated as better positions found by other particles. Such a strategy is expected to move the swarm towards the best solutions.

Recent researches (R.-B. Chen et al., 2015; R.-B. Chen et al., 2013; Wong et al., 2015) adapted the classic PSO algorithm to search optimal experimental designs, which is a discrete optimization task. Specifically, R.-B. Chen et al., 2013 proposed a PSO based algorithm, denoted as the LaPSO, to identify maximin distance LHDs. It re-defines the steps on how each particle updates its velocity and position in the general PSO framework. It targets on reducing the Hamming distances between each particle and the global best (or corresponding personal best) via exchanging elements in a deterministic way. The Hamming distance between two particle LHDs is the total number of positions whose corresponding elements are different.

LaPSO starts from generating m random LHDs, where the number of particles m is a tuning parameter. At the beginning, the personal best (PB) of each particle LHD is itself. The global best (GB) is the particle LHD who has the best performance in terms of a given optimality criterion Φ. Next, an exchange procedure is implemented for each column in each particle LHD. That is, the algorithm randomly chooses an element in the current column (denoted as e_r) and finds another element in the corresponding PB whose location is the same as e_r (denoted as e_p). In the current column, exchange e_r with e_p. Such an exchange reduces the Hamming distance between the current particle LHD and its PB. Similarly, the algorithm implements the exchange procedure to reduce the Hamming distance between the current particle LHD and the GB. These two steps mimic how each particle updates its position and moves towards the PB and the GB. Either of these two steps can be skipped or repeated, but they cannot be skipped simultaneously. Though there is no hard limit on the number of repetitions, a large number, e.g. $2n$, will make every element in the current column nearly the same as the PB or the GB. After all particle LHDs have been updated, for each column in each particle LHD, draw a random number z from the standard Uniform distribution. If $z < p0$, where $p0$ is an user-defined tuning parameter, two randomly selected elements will be exchanged in the current column. Such a procedure aims to prevent the algorithm from getting stuck at a local optima. After convergence or reaching the budget constraint, the GB would be returned. Different from the exchange procedure in the SA based algorithms, LaPSO

connects each particle with the personal best, the global best or both. As every exchange step at most reduces the Hamming distance by one, the LaPSO may be computationally cumbersome for identifying large optimal designs. The pseudo-code for our implementation of LaPSO is outlined in Algorithm 2. In the `LHD` package, function `LaPSO()` implements this algorithm.

Algorithm 2 LaPSO

Generate m random LHDs, denoted as $L_1, \ldots, L_m$.
Initialize the personal best. Set $PB_i = L_i$ for $i = 1, \ldots, m$.
Initialize the global best. Set $GB = \underset{i}{\operatorname{argmin}}\, \Phi(L_i)$.
while not converge **do**
  for each particle LHD L_i **do**
    for each column j of L_i **do**
      Randomly choose an element in column j, denoted as e_r, and then find the element in PB_i whose location is same as e_r, denoted that as e_p. Swap e_r and e_p within the column j.
      Randomly choose an element in column j, denoted as e_r, and then find the element in the GB whose location is same as e_r, denoted that as e_g. Swap e_r and e_g within the column j.
      if $z < p_0$, where $z \sim$ Uniform(0,1) **then**
        Exchange two randomly selected elements in column j.
      end if
    end for
  end for
  for each updated L_i **do**
    if $\Phi(L_i) < \Phi(PB_i)$ **then**
      $PB_i = L_i$.
    end if
    if $\Phi(L_i) < \Phi(GB)$ **then**
      $GB = L_i$.
    end if
  end for
end while

2.3.3 Genetic Algorithms

Genetic algorithm (GA) is a nature-inspired meta-heuristic optimization algorithm which mimics Charles Darwin's idea of natural selection (Goldberg, 1989; J. H. Holland et al., 1992). GA is a population-based algorithm which generally includes the steps of *selection*, *crossover*, *mutation* and *fitness*. Following Darwin's terminology, candidate solutions in GA are called chromosomes, which form a population. Based on the quality of performance (called *fitness*) for a given measure, several chromosomes will be chosen as parents for generating new population, which is called *selection*. Parents will be combined with each other using the methods of *crossover* and *mutation* to produce offspring that have, hopefully, better performance.

Liefvendahl and Stocki, 2006 proposed a GA for identifying maximin distance LHDs. The initial population is formed by m random LHDs, where an even number m is an user-defined population size. In the *selection* step, the entire population will be evaluated based on the given optimality criterion and the best half will be selected as *survivors*. The current global best will be marked as the *best survivor* (BS). Then, BS and other *survivors* will be used for generating a new population. Then, let BS be the first individual and generate the 2nd to the $(m/2)$th individuals by the following *crossover* step. For each *survivor* except the BS, randomly choose a column in the current *survivor* and replace this column of BS with a randomly selected column to generate a new individual. For the other half of the new population, let BS be the $(m/2+1)$th individual and generate the $(m/2+2)$th to the mth individuals by the following *crossover* step. For each *survivor* except the BS, randomly choose a column (e.g. column j) in the current *survivor* and replace it with the jth column in BS to produce a new individual. After forming such a new population of m new LHDs, the *mutation* step is used to avoid being trapped at local optima. In the *mutation* step, for every column in all new LHDs except for the first design, draw a random number z from the standard Uniform distribution, and if $z < p_{mut}$, where p_{mut} is an user-defined tuning parameter, two randomly selected elements in this column will be exchanged. Next, the entire population will be evaluated again to determine the new *survivors* for the next iteration. After the algorithm converges or the budget constraint is reached, the BS would be returned. Rather than exchanging elements in the LaPSO, the GA replaces entire columns in the current "global best" design as well as the candidate designs. In GA, the Hamming distance for the replaced columns becomes zero immediately towards the global best. Compared to other search algorithms, it generally requires less CPU time, especially for large designs sizes. The pseudo-code of GA is outlined in Algorithm 3. In the `LHD` package, function `GA()` implements this algorithm.

2.3.4 Illustrating Examples for the Implemented Search Algorithms

This subsection demonstrates some examples on how to use the search algorithms in the developed `LHD` package. Recommendations of input arguments and tuning parameters are provided for practitioners.

To generate a random LHD matrix, function `rLHD` can be used, where input arguments are the run-size `n` and the factor-size `k`.

```
> X=rLHD(n=5,k=3);X  #This generates a 5 by 3 random LHD, denoted as X
     [,1] [,2] [,3]
[1,]    2    1    4
[2,]    4    3    3
[3,]    3    2    2
[4,]    1    4    5
[5,]    5    5    1
```

Next, we evaluate the random LHD in terms of different optimality criteria. Note that the default settings for `phi_p` is $p = 15$ and $q = 1$ (the Manhattan distance) and user can change the settings.

```
> phi_p(X)              #The maximin L1-distance criterion.
[1] 0.3336608
```

Algorithm 3 Genetic Algorithm for LHD

1: Generate m random LHDs, denoted as $L_1, \ldots, L_m$, where m is an even number.
2: Calculate $\Phi(L_i)$ for $i = 1, \ldots, m$, and select the best $\frac{m}{2}$ L_i (with the smallest $\frac{m}{2}$ Φ values), denoted by L_i^s for $i = 1, \ldots, \frac{m}{2}$, WLOG.
3: Identify the *best survivor*, L_b^s, i.e. $L_b^s = \underset{i}{\operatorname{argmin}}\, \Phi(L_i^s)$.
4: **while** not converge **do**
5: Let $L_1 = L_b^s$ and $c = 2$.
6: **for** each L_i^s (except L_b^s) **do**
7: Randomly choose a column j from L_i^s, and use it to replace the jth column of L_b^s. Let this new matrix be L_c, and $c = c + 1$.
8: **end for**
9: Let $L_{m/2+1} = L_b^s$ and $c = m/2 + 2$.
10: **for** each L_i^s (except L_b^s) **do**
11: Randomly choose a column j from L_i^s, and replace it with the j^{th} column from L_b^s. Let this new matrix be L_c, and $c = c + 1$.
12: **end for**
13: **for** each L_i (except L_1) **do**
14: **for** each column j of L_i **do**
15: **if** $z < p_{mut}$, where $z \sim \text{Uniform}(0, 1)$ **then**
16: Exchange two randomly selected elements in column j.
17: **end if**
18: **end for**
19: **end for**
20: Repeat Steps $2 - 3$.
21: **end while**

Table 2.2: Search algorithm functions in the LHD package

Function	Description
SA	Returns an LHD via the simulated annealing algorithm (Morris & Mitchell, 1995).
OASA	Returns an LHD via the orthogonal-array-based simulated annealing algorithm (Leary et al., 2003), where an OA of the required design size must exist.
SA2008	Returns an LHD via the simulated annealing algorithm with the multi-objective optimization approach (Joseph & Hung, 2008).
SLHD	Returns an LHD via the improved two-stage algorithm by Ba et al., 2015.
LaPSO	Returns an LHD via the particle swarm optimization (R.-B. Chen et al., 2013).
GA	Returns an LHD via the genetic algorithm (Liefvendahl & Stocki, 2006).

```
> phi_p(X,p=10,q=2)     #The maximin L2-distance criterion.
[1] 0.5797347
> MaxProCriterion(X)    #The Maximum Projection Criterion.
[1] 0.5375482
> AvgAbsCor(X)          #Average Absolute Correlation.
[1] 0.5333333
> MaxAbsCor(X)          #Maximum Absolute Correlation.
[1] 0.9
```

In Table 2.2, we summarize the R functions of the six algorithms discussed in the previous subsections, which can be used to identify different types of optimal LHDs. Users who seek fast solutions can use the default settings of input arguments after specifying the design sizes. See the following examples.

```
#Generate a 5 by 3 maximin distance LHD by the SA function.
> try.SA=SA(n=5,k=3); try.SA
     [,1] [,2] [,3]
[1,]    2    2    1
[2,]    5    3    2
[3,]    4    5    5
[4,]    3    1    4
[5,]    1    4    3
> phi_p(try.SA)   #\phi_p is smaller than that of a random LHD (0.3336608).
[1] 0.2169567

#Generates a 5 by 3 maximin distance LHD by the SA2008, SLHD, LaPSO
# and GA functions.
> try.SA2008=SA2008(n=5,k=3)
> try.SLHD=SLHD(n=5,k=3)
```

```
> try.LaPSO=LaPSO(n=5,k=3)
> try.GA=GA(n=5,k=3)

#Generate an OA(9,2,3,2).
> OA=matrix(c(rep(1:3,each=3),rep(1:3,times=3)),
+           ncol=2,nrow=9,byrow = FALSE)
#Generates a maximin distance LHD with the same design size as the input OA
#by the orthogonal-array-based simulated annealing algorithm.
> try.OASA=OASA(OA)
>OA; try.OASA
      [,1] [,2]         [,1] [,2]
[1,]    1    1    [1,]    1    2
[2,]    1    2    [2,]    2    6
[3,]    1    3    [3,]    3    9
[4,]    2    1    [4,]    4    3
[5,]    2    2    [5,]    6    5
[6,]    2    3    [6,]    5    7
[7,]    3    1    [7,]    7    1
[8,]    3    2    [8,]    9    4
[9,]    3    3;   [9,]    8    8
```

Note that the default optimality criterion embedded in all search algorithms is "phi_p" (i.e. the maximin distance criterion), which lead to the maximin L_2-distance LHDs. For other optimality criteria, users should change the setting of the input argument `OC` (with options "phi_p", "MaxProCriterion", "MaxAbsCor" and "MaxProCriterion"). The following examples illustrate some details on different argument settings.

```
#Design try.SA is a 5 by 3 maximin distance LHD generated by the SA with
#30 iterations. The temperature starts at 10 and decreases 10% each time.
#The minimium temperature allowed is 1 and the maximum perturbations that
#the algorithm will try without improvements is 5. The optimality criterion
#used is \phi_p with p=15 and q=1, and the maximum CPU time is 5 minutes.
> try.SA=SA(n=5,k=3,N=30,T0=10,rate=0.1,Tmin=1,Imax=5,OC="phi_p",
+           p=15,q=1,maxtime=5); try.SA
      [,1] [,2] [,3]
[1,]    1    3    4
[2,]    2    5    2
[3,]    5    4    3
[4,]    4    1    5
[5,]    3    2    1
> phi_p(try.SA)
[1] 0.2169567
```

```
#try.SA2008 below is a 5 by 3 maximin distance LHD generated by SA with
#the multi-objective optimization approach.
> try.SA2008=SA2008(n=5,k=3,N=30,T0=10,rate=0.1,Tmin=1,Imax=5,
+                   OC="phi_p",p=15,q=1,maxtime=5)

#try.SLHD below is a 5 by 3 maximin distance LHD generated by the improved
#two-stage algorithm with only 1 slice (t=1).
> try.SLHD=SLHD(n=5,k=3,t=1,N=30,T0=10,rate=0.1,Tmin=1,Imax=5,
+               OC="phi_p",p=15,q=1,stage2=TRUE,maxtime=5)

#try.OASA below is a 9 by 2 maximin distance LHD generated by the
#orthogonal-array-based simulated annealing algorithm with the input
#OA (defined previously) and the rest arguments are interpreted the
#same as the design try.SA above.
> try.OASA=OASA(OA,N=30,T0=10,rate=0.1,Tmin=1,Imax=5,OC="phi_p",p=15,
+               q=1,maxtime=5)

#try.LaPSO below is a 5 by 3 maximum projection LHD generated by the
#particle swarm optimization algorithm with 20 particles and 30
#iterations. Zero (or two) elements in any column of the current particle
#should be the same as the corresponding personal best (or global best).
#The probability of exchanging two randomly selected elements is 0.5,
#and the maximum CPU time is 5 minutes.
> try.LaPSO=LaPSO(n=5,k=3,m=20,N=30,SameNumP=0,SameNumG=2,p0=0.5,
+                 OC="MaxProCriterion",maxtime=5); try.LaPSO
[,1] [,2] [,3]
[1,]    4    5    4
[2,]    3    1    3
[3,]    5    2    1
[4,]    2    3    5
[5,]    1    4    2
#Compare to the value of 0.5375482 from the above random LHD.
> MaxProCriterion(try.LaPSO)
[1] 0.3561056

#try.GA below is a 5 by 3 OLHD generated by the genetic algorithm
#with the population size 20, number of iterations 30,  mutation
#probability 0.5 and maximum CPU time 5 minutes.
> try.GA=GA(n=5,k=3,m=20,N=30,pmut=0.5,OC="MaxAbsCor",maxtime=5)
```

Next, we discuss some implementation details. In the SA based algorithms (`SA`, `SA2008`, `SLHD` and `OASA`), the number of iterations `N` is recommended to be no greater than 500 according to the convergence

curves in Figure 2 of Section 5. Input `rate` determines the percent of the current temperature decreased (e.g. 0.1 means a decrease of 10% each time). A high rate would make the temperature decline quickly, which leads to a fast stop of the algorithm. It is recommended to set `rate` from 0.1 to 0.15. `Imax` indicates the maximum perturbations that the algorithm will try without improvements before the temperature reduces, and it is recommended to be no greater than 5 for computing time considerations. `OC` chooses the optimality criterion and is the `"phi_p"` criterion in (2.1) by default. `OC` has other options, including `"MaxProCriterion"`, `"AvgAbsCor"` and `"MaxAbsCor"`. The function `SLHD` has two additional input arguments, which are `t` and `stage2`. Here `t` is the number of slices in the design. `SLHD` can be used as a modified SA algorithm when `t` is 1. `stage2` is a logical argument (either `TRUE` or `FALSE`) which determines if the second stage of the algorithm would be implemented, and is recommended to set as `FALSE` when computational resources are limited.

For the function `LaPSO`, the input `m` is the number of particles. There are three tuning parameters: `SameNumP`, `SameNumG` and `p0`. `SameNumP` and `SameNumG` denote how many exchanges would be performed to reduce the Hamming distance towards the personal best and the global best. `p0` denotes the probability of a random swap for two elements in the current column of the current particle. In R.-B. Chen et al., 2013, they provided the following suggestions: `SameNumP` is approximately $n/2$ when `SameNumG` is 0, `SameNumG` is approximately $n/4$ when `SameNumP` is 0, and `p0` should be between $1/(k-1)$ and $2/(k-1)$. For function `GA`, input `m` is the population size. The only tuning parameter, i.e. the mutation probability `pmut`, is recommended to be $1/(k-1)$. For every algorithm, we incorporate a progress bar for visualizing the computing time used. After an algorithm completes, information of "average CPU time per iteration" and "numbers of iterations completed" would be presented. Users can set the limit for the CPU time used for each algorithm via the argument `maxtime`. Our algorithms support both the L_1 and L_2 distances.

2.4 Algebraic Constructions for Optimal LHDs with Certain Sizes

For some design sizes, algebraic constructions are available and theoretical results are developed to guarantee the efficiency of such designs. Algebraic constructions require nearly no searching, and thus are especially attractive for large designs. In this section, we review some practically useful algebraic constructions for maximin distance LHDs and orthogonal LHDs.

2.4.1 Algebraic Constructions for Maximin Distance LHDs

L. Wang et al., 2018 proposed to generate maximin distance LHDs via good lattice point (GLP) sets (Zhou & Xu, 2015) and Williams transformation (E. Williams, 1949). In practice, their method can lead to space-filling designs with relatively flexible sizes, where the run size n is flexible but the factor size k must be no greater than the number of positive integers that are co-prime to n. They proved that the resulting designs of sizes $n \times (n-1)$ (with n being any odd prime) and $n \times n$ (with $2n+1$ or $n+1$ being odd prime) are optimal under the maximin L_1 distance criterion.

The construction method in L. Wang et al., 2018 can be summarized into three steps. First, generate an $n \times k$ GLP design D whose element is $x_{ij} = i \times h_j \pmod n$ with $i = 1, \ldots, n$, $j = 1, \ldots, k$ and $h = (h_1, \ldots, h_k)$ being a set of distinct positive integers that are coprime to n. Second, for any $b \in \{0, \ldots, n-1\}$, generate $D_b = D + b \pmod n$ and $E_b = W(D_b)$, where $W : \mathcal{Z}_n \to \mathcal{Z}_n$ is the Williams transformation (E. Williams, 1949):

$$W(x) = \begin{cases} 2x, & 0 \le x \le (n-1)/2, \\ 2(n-x)-1, & (n+1)/2 \le x \le n-1. \end{cases} \tag{2.4}$$

Third, find the best $b^* \in \{0, \ldots, n-1\}$ such that E_{b^*} has the smallest ϕ_p value.

By Zhou and Xu, 2015 and L. Wang et al., 2018, designs D and E defined above are $n \times k$ LHDs with the last rows having the same elements. We can remove their last rows, and re-order their levels such that the remaining designs are LHDs, denoted as the Leave-one-out designs $\tilde{D}$ and $\tilde{E}$. After performing the three steps outlined above, the resulting designs E_{b^*} and $\tilde{E}_{b^*}$ have good space-filling properties. This construction method by L. Wang et al., 2018 is very attractive for constructing large maximin distance LHDs, but may be too complex for practitioners to implement in practice. In the `LHD` package, function `FastMmLHD()` implements this method with an user-friendly setup.

Tang, 1993 proposed to construct orthogonal array-based LHDs (OALHDs) from existing orthogonal arrays (OAs). Suppose $\mathbf{A}$=OA(n, m, s, r) is an orthogonal array with n rows, m columns, s levels ($n > s \ge 2$) and r strength. If each $n \times r$ sub-matrix of $\mathbf{A}$ includes all possible $1 \times r$ row vectors with the same frequency λ. Then, λ is called the index of the array where the run size $n = \lambda s^r$. An $n \times m$ LHD can be considered as a OA$(n, m, n, 1)$ with $\lambda = 1$. For every column of an OA, Tang, 1993 proposed to replace the ns^{-1} (equal to λs^{r-1}) positions of entry k by a permutation of numbers: $(k-1)ns^{-1} + 1, (k-1)ns^{-1} + 2, \ldots, (k-1)ns^{-1} + ns^{-1} = kns^{-1}$, where $k = 1, \ldots, s$. Then, the new generated design matrix would be an $n \times m$ LHD, which is called an orthogonal array-based LHD (OALHD). Tang, 1993 showed that the OALHDs can have better space-filling properties than the general ones. In the `LHD` package, function `OA2LHD()` implements this method.

2.4.2 Algebraic Constructions for Orthogonal LHDs

Orthogonal LHDs (OLHDs) have zero pairwise correlation between any two columns, which are widely used by practitioners. There is rich literature on the constructions of OLHDs with various design sizes, but they are often too hard for practitioners to replicate in practice. In this part, we summarize and implement some currently popular methods (Butler, 2001; Cioppa & Lucas, 2007; C. D. Lin et al., 2009; Sun et al., 2010; Tang, 1993; Ye, 1998). We summarize their design sizes in Table 2.3.

Ye, 1998 proposed a construction for OLHDs with run sizes $n = 2^m + 1$ and factor sizes $k = 2m - 2$ where m is any integer bigger than 2. It involves the constructions of three matrices $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{T}$. The first column in $\mathbf{M}$, denoted as $\mathbf{e}$, is a random permutation of $(1, 2, \ldots, 2^{m-1})$. The 2nd to the mth columns

Table 2.3: Summary Table of Run and Factor Sizes for Different Construction Methods.

	Ye98	Cioppa07	Sun10	Lin09	Butler01
Run (n)	$2^m + 1$	$2^m + 1$	$r2^{c+1}$ or $r2^{c+1} + 1$	n^2	n
Factor (k)	$2m - 2$	$m + \binom{m-1}{2}$	2^c	$2fp$	$k \leq n - 1$
Note	m is a positive integer, $m \geq 2$	m is a positive integer, $m \geq 2$	r and c are positive integers	n^2, $2f$ and p are from OA($n^2, 2f, n, 2$) and OLHD(n,p)	n is an odd prime number

in **M** are calculated as $\mathbf{A}_L\mathbf{e}$, where $L = 1, 2, \ldots, m-1$. Each $\mathbf{A}_L$ is defined as:

$$\mathbf{A}_L = \underbrace{\mathbf{I} \otimes \ldots \otimes \mathbf{I}}_{m-1-L} \otimes \underbrace{\mathbf{R} \otimes \ldots \otimes \mathbf{R}}_{L},$$

where **I** is the 2×2 identity matrix, $\mathbf{R} = \mathbf{1}\mathbf{1}^T - \mathbf{I}$ and $\otimes$ is the Kronecker product. The last $m-2$ columns in **M** are calculated as $\mathbf{A}_i\mathbf{A}_{m-1}\mathbf{e}$, where $i = 1, 2, \ldots, m-2$. Similarly, to construct the matrix **S**, set its first column, denoted as **j**, to be the $2^{m-1} \times 1$ vector of $+1$'s. The 2nd to the mth columns in **S**, denoted as $\mathbf{a}_K$ with $K = 1, 2, \ldots, m-1$ are defined as $\mathbf{a}_K = \mathbf{B}_1 \otimes \mathbf{B}_2 \otimes \ldots \otimes \mathbf{B}_{m-1}$, where all the **B**'s are $[1, 1]^T$ except for $\mathbf{B}_{m-k} = [-1, 1]^T$. The last $m-2$ columns in **S** are calculated as $\mathbf{a}_1\mathbf{a}_j$, where $j = 2, \ldots, m-1$. Matrix **T** is the element-wise product of **M** and **S**, and the matrix **X** is constructed by $\mathbf{X} = [\mathbf{T}^T, \mathbf{o}, \text{-}\mathbf{T}^T]^T$, where **o** is a column vector containing k zeros and $\text{-}\mathbf{T} = -1 \times \mathbf{T}$. For any choice of **e** in **M**, the resulting design **X** is an OLHD. In the `LHD` package, function `OLHD.Y1998()` implements this algebraic construction.

Cioppa and Lucas, 2007 extended Ye, 1998's method to construct OLHDs with run size $n = 2^m + 1$ and factor size $k = m + \binom{m-1}{2}$, where m is any integer bigger than 2. Their construction adopted the same matrix **T** in Ye, 1998 but used different matrices **M** and **S**. In **M**, for the last $m-2$ columns, instead of using $\mathbf{A}_i\mathbf{A}_{m-1}\mathbf{e}$ in Ye, 1998, they adopted $\mathbf{A}_i\mathbf{A}_j\mathbf{e}$ with $i = 1, \ldots, m-2$ and $j = i+1, \ldots, m-1$, which accommodates more factors. Similarly, for the matrix **S**, for the last $m-2$ columns, they adopted $\mathbf{a}_i\mathbf{a}_j$ with $i = 1, \ldots, m-2$ and $j = i+1, \ldots, m-1$. Given the same number of runs, the method in Cioppa and Lucas, 2007 is capable of accommodating more factors. Note that the choice of **e** in constructing **M** is important. If $\mathbf{e} = [1, 2, \ldots, 2^{m-1}]^T$, then **X** is guaranteed to be orthogonal, while a poor choice of **e** may not guarantee **X** to be orthogonal. In the `LHD` package, function `OLHD.C2007()` implements this algebraic construction.

Sun et al., 2010 extended their earlier work (Sun et al., 2009) to construct OLHDs with $n = r2^{c+1} + 1$ or $n = r2^{c+1}$ and $k = 2^c$, where r and c are positive integers. Their method for constructing OLHD

with $n = r2^{c+1} + 1$ and $k = 2^c$ consists of constructing three matrices: S_c, T_c and $A_{r2^c \times 2^c}$. The matrix S_c is defined as

$$S_c = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \text{ for } c = 1, S_c = \begin{bmatrix} S_{c-1} & -S^*_{c-1} \\ S_{c-1} & S^*_{c-1} \end{bmatrix} \text{ for } c > 1,$$

where operator $*$ works on any matrix with an even number of rows by multiplying the top half entries by -1. The matrix T_c is defined as

$$T_c = \begin{cases} \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix} & \text{for} \quad c = 1, \\ \begin{bmatrix} T_{c-1} & -(T^*_{c-1} + 2^{c-1} S^*_{c-1}) \\ T_{c-1} + 2^{c-1} S_{c-1} & T^*_{c-1} \end{bmatrix} & \text{for} \quad c > 1. \end{cases}$$

The matrix $A_{r2^c \times 2^c}$ is defined as $A_{r2^c \times 2^c} = [(T^1_c)^T, \ldots, (T^r_c)^T]^T$, where $T^i_c = T_c + (i-1)2^c S_c$ for $i = 1, \ldots, r$. The design matrix $\mathbf{X}$ is given by $\mathbf{X} = [A^T_{r2^c \times 2^c}, \mathbf{o}, -A^T_{r2^c \times 2^c}]^T$, where $\mathbf{o}$ is a column vector containing 2^c zeros. The construction method for OLHD with $n = r2^{c+1}$ and $k = 2^c$ also relies on the S_c and T_c. Let $H_c = T_c - S_c/2$, $B_{r2^c \times 2^c}$ is defined as $B_{r2^c \times 2^c} = [(H^1_c)^T, \ldots, (H^r_c)^T]^T$, where $H^i_c = H_c + (i-1)2^c S_c$ for $i = 1, \ldots, r$. The design matrix $\mathbf{X}$ is constructed as $\mathbf{X} = [B^T_{r2^c \times 2^c}, -B^T_{r2^c \times 2^c}]^T$. The proposed method is flexible in terms of run sizes as they can be either even or odd. It accommodates more factors than that in Ye, 1998 but the factor sizes are restricted to be powers of two. In the `LHD` package, function `OLHD.S2010()` implements this algebraic construction.

C. D. Lin et al., 2009 constructed OLHDs and NOLHDs with n^2 runs and $2fp$ factors by coupling OAs. In practice, it starts with an OLHD(n, p) or NOLHD(n, p), which will be coupled with an OA($n^2, 2f, n, 2$). Let b_{ij} denote the elements of OLHD(n, p), where $i = 1, \ldots, n$ and $j = 1, \ldots, p$, and $\mathbf{A}$=OA($n^2, 2f, n, 2$). For $j = 1, \ldots, p$, construct an $n^2 \times (2f)$ matrix A_j from $\mathbf{A}$ by replacing its levels $1, \ldots, n$ with $b_{ij}, \ldots, b_{nj}$. Partition each A_j matrix into f pieces, e.g. $A_j = [A_{j1}, \ldots, A_{jf}]$, where each of $A_{j1}, \ldots, A_{jf}$ contains exactly two columns. Let $M_j = [A_{j1}V, \ldots, A_{jf}V]$, where

$$V = \begin{bmatrix} 1 & -n \\ n & 1 \end{bmatrix},$$

and the design matrix $\mathbf{X} = [M_1, \ldots, M_p]$. For example, an OLHD(11, 7), coupled with an OA(121,12,11,2), would yield an OLHD(121, 84). Similarly, a NOLHD(169, 168) can be generated from an NOLHD(13, 12) coupled with an OA(169,14,13,2). One advantage of their construction method is that it requires fewer runs to accommodate a large number of factors. The number of factors here can only be even numbers and the run size is restricted by the availability of the OAs. In the `LHD` package, function `OLHD.L2009()` implements this algebraic construction.

Butler, 2001 proposed a method to construct OLHDs with the run size n being odd primes via the Williams transformation (E. Williams, 1949). When the factor size $k \leq (n-1)/2$, let Y denote an $n \times k$

matrix and the elements of Y are defined as

$$y_{ij} = \begin{cases} ig_j + (n-1)/4 \bmod n, & \text{for} \quad n = 1, 5, 9, \ldots, \quad i.e., n \equiv 1 \bmod 4, \\ ig_j + (3n-1)/4 \bmod n, & \text{for} \quad n = 3, 7, 11, \ldots, \quad i.e., n \equiv 3 \bmod 4, \end{cases}$$

where $y_{ij} \in \{0, 1, \ldots, n-1\}, i = 1, 2, \ldots, n, j = 1, 2, \ldots, k$, and $g_1, g_2, \ldots, g_k$ are distinct elements in the set $\{1, 2, \ldots, (n-1)/2\}$. The design matrix would be $\mathbf{X} = W(Y)$, where the Williams transformation W is defined by Equation (2.4). When $(n-1)/2 + 1 \leq k \leq (n-1)$, let $r = k - (n-1)/2$ and the design matrix would be partitioned as $\mathbf{X} = (\mathbf{X}_0, \mathbf{X}_1)$, where $\mathbf{X}_0$ is the $n \times (n-1)/2$ design matrix generated by the same procedure as above and $\mathbf{X}_1$ is an $n \times r$ design matrix having the following elements before applying the Williams transformation:

$$x_{ij} \equiv ig_j \bmod n,$$

where $x_{ij} \in \{0, 1, \ldots, n-1\}, i = 1, 2, \ldots, n, j = 1, 2, \ldots, r$, and $g_1, g_2, \ldots, g_r$ are distinct elements in the set $\{1, 2, \ldots, (n-1)/2\}$. Note that n can be any odd number in theory, but it is often assumed to be a prime number so that the generated designs are LHDs. In the `LHD` package, function `OLHD.B2001()` implements this algebraic construction.

2.4.3 Illustrating Examples for the Implemented Algebraic Constructions

In Table 2.4, we summarize the algebraic construction methods implemented by the developed `LHD` package, where `FastMmLHD` and `OA2LHD` are for maximin distance LHDs and `OLHD.Y1998`, `OLHD.C2007`, `OLHD.S2010`, `OLHD.L2009` and `OLHD.B2001` are for orthogonal LHDs. The following examples will illustrate how to use them.

```
#FastMmLHD(8,8) generates an optimal 8 by 8 maximin L_1 distance LHD.
>try.FastMm=FastMmLHD(n=8,k=8); try.FastMm
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0    1    2    3    4    5    6    7
[2,]    1    3    5    7    6    4    2    0
[3,]    2    5    7    4    1    0    3    6
[4,]    3    7    4    0    2    6    5    1
[5,]    4    6    1    2    7    3    0    5
[6,]    5    4    0    6    3    1    7    2
[7,]    6    2    3    5    0    7    1    4
[8,]    7    0    6    1    5    2    4    3

#OA2LHD(OA) expands an input OA to an LHD of the same run size.
>try.OA2LHD=OA2LHD(OA)
>OA; try.OA2LHD
     [,1] [,2]          [,1] [,2]
[1,]    1    1     [1,]    1    2
```

Table 2.4: Algebraic construction methods in the LHD package

Function	Description
FastMmLHD	Returns a maximin distance LHD matrix (L. Wang et al., 2018).
OA2LHD	Expands an orthogonal array to an LHD (Tang, 1993).
OLHD.Y1998	Returns a $2^m + 1$ by $2m - 2$ orthogonal LHD matrix (Ye, 1998) where m is an integer and $m \geq 2$.
OLHD.C2007	Returns a $2^m + 1$ by $m + \binom{m-1}{2}$ orthogonal LHD matrix (Cioppa & Lucas, 2007) where m is an integer and $m \geq 2$.
OLHD.S2010	Returns a $r2^{c+1} + 1$ or $r2^{c+1}$ by 2^c orthogonal LHD matrix (Sun et al., 2010) where r and c are positive integers.
OLHD.L2009	Couples an n by p orthogonal LHD with a n^2 by $2f$ strength 2 and level n orthogonal array to generate a n^2 by $2fp$ orthogonal LHD (C. D. Lin et al., 2009).
OLHD.B2001	Returns an orthogonal LHD (Butler, 2001) with the run size being odd primes.

```
[2,]    1    2     [2,]    2    4
[3,]    1    3     [3,]    3    9
[4,]    2    1     [4,]    4    3
[5,]    2    2     [5,]    5    5
[6,]    2    3     [6,]    6    7
[7,]    3    1     [7,]    9    1
[8,]    3    2     [8,]    8    6
[9,]    3    3;    [9,]    7    8

#OLHD.Y1998(m=3) generates a 9 by 4 orthogonal LHD.
#Note that 2^m+1=9 and 2*m-2=4.
> try.Y1998=OLHD.Y1998(m=3);try.Y1998
      [,1] [,2] [,3] [,4]
[1,]    4   -3   -2    1
[2,]    3    4   -1   -2
[3,]    1   -2    3   -4
[4,]    2    1    4    3
[5,]    0    0    0    0
[6,]   -4    3    2   -1
[7,]   -3   -4    1    2
[8,]   -1    2   -3    4
```

```
[9,]   -2   -1   -4   -3
> MaxAbsCor(try.Y1998)    #column-wise correlations are 0.
[1] 0

#OLHD.C2007(m=4) generates a 17 by 7 orthogonal LHD.
#Note that 2^m+1=17 and $4+{4-1 \choose 2}$=7.
> try.C2007=OLHD.C2007(m=4); dim(try.C2007)
[1] 17  7
> MaxAbsCor(try.C2007)    #column-wise correlations are 0
[1] 0

#OLHD.S2010(C=3,r=3,type="odd") generates a 49 by 8 orthogonal LHD.
#Note that 3*2^4+1=49 and 2^3=8.
> dim(OLHD.S2010(C=3,r=3,type="odd"))
[1] 49  8
> MaxAbsCor(OLHD.S2010(C=3,r=3,type="odd")) #column-wise correlations are 0
[1] 0

#OLHD.S2010(C=3,r=3,type="even") generates a 48 by 8 orthogonal LHD.
#Note that 3*2^4=48 and 2^3=8.
> dim(OLHD.S2010(C=3,r=3,type="even"))
[1] 48  8
> MaxAbsCor(OLHD.S2010(C=3,r=3,type="even")) #column-wise correlations are 0
[1] 0

#Create a 5 by 2 OLHD.
OLHD=OLHD.C2007(m=2)

#Create an OA(25,6,5,2).
OA=matrix(c(2,2,2,2,2,1,2,1,5,4,3,5,3,2,1,5,4,5,1,5,4,3,2,5,4,1,3,5,2,3,
1,2,3,4,5,2,1,3,5,2,4,3,1,1,1,1,1,1,4,3,2,1,5,5,5,5,5,5,5,1,4,4,4,4,4,1,
3,1,4,2,5,4,3,3,3,3,3,1,3,5,2,4,1,3,3,4,5,1,2,2,5,4,3,2,1,5,2,3,4,5,1,2,
2,5,3,1,4,4,1,4,2,5,3,4,4,2,5,3,1,4,2,4,1,3,5,3,5,3,1,4,2,4,5,2,4,1,3,3,
5,1,2,3,4,2,4,5,1,2,3,2),ncol=6,nrow=25,byrow=TRUE)

#OLHD.L2009(OLHD,OA) generates a 25 by 12 orthogonal LHD.
#Note that n=5 so n^2=25. p=2 and f=3 so 2fp=12.
> dim(OLHD.L2009(OLHD,OA))
[1] 25 12
> MaxAbsCor(OLHD.L2009(OLHD,OA))    #column-wise correlations are 0.
[1] 0
```

Table 2.5: Minimum ϕ_p and Average CPU Time (in seconds) under the L_1 Distance.

$n \times k$	SA	SA2008	SLHD	LaPSO	GA	FastMm
6×6	0.0874(10)	0.0874(54)	0.0883(15)	0.0856(26)	0.0856(9)	0.0856(0)
7×6	0.0816(13)	0.0776(57)	0.0803(20)	0.0777(32)	0.0766(11)	0.0766(0)
8×8	0.0555(26)	0.0529(127)	0.0545(37)	0.0526(62)	0.0524(19)	0.0520(0)
9×9	0.0455(35)	0.0432(165)	0.0447(50)	0.0426(82)	0.0425(24)	0.0423(0)
10×10	0.0380(39)	0.0358(153)	0.0380(56)	0.0354(87)	0.0353(25)	0.0353(0)
11×10	0.0361(47)	0.0333(166)	0.0359(67)	0.0331(103)	0.0329(29)	0.0327(0)
12×12	0.0280(70)	0.0260(290)	0.0279(99)	0.0256(157)	0.0256(43)	0.0258(0)
13×12	0.0263(92)	0.0245(351)	0.0264(132)	0.0242(205)	0.0240(55)	0.0240(0)
14×14	0.0212(119)	0.0197(467)	0.0211(169)	0.0194(261)	0.0194(69)	0.0193(0)

```
#OLHD.B2001(n=11,k=5) generates a 11 by 5 orthogonal LHD.
> dim(OLHD.B2001(n=11,k=5))
[1] 11  5
```

2.5 Numerical Results and Comparisons

In this section, we aim to provide suggestions for practitioners on how to choose appropriate method(s) for generating optimal LHDs. Simulations are conducted to compare different search algorithms and algebraic constructions. All methods are implemented by our developed R package `LHD`. For each design size, we run every search algorithm 20 times and record the best design found and the average CPU time. In all tables, we write a "0" if the CPU time was smaller than one second, and we set the maximum number of iterations to be 500 for Algorithms 1, 2 and 3 in Section 2.3.

2.5.1 Results on Maximin Distance LHDs

First, we compare the five search algorithms: SA (Morris & Mitchell, 1995), SA2008 (Joseph & Hung, 2008), SLHD (Ba et al., 2015), LaPSO (R.-B. Chen et al., 2013) and GA (Liefvendahl & Stocki, 2006) along with the algebraic method FastMm (L. Wang et al., 2018) for constructing Maximin distance LHDs under both the L_1 and L_2 distances. All these methods and their implementation details using the R package `LHD` are demonstrated in Sections 2.3 and 2.4. We discuss the cases under the L_1 distance in Table 2.5 and the cases under the L_2 distance in Tables 2.6 and 2.7. Note that the Maximin L_1 distance designs may be very different from the Maximin L_2 distance designs when the design sizes are small. For large design sizes, they tend to be similar.

Table 2.6: Minimum ϕ_p and Average CPU Time (in seconds) with Run Size Between 4 and 13 under the L_2 Distances.

	k	2	3	4
n		Min(CPU)	Min(CPU)	Min(CPU)
4	SA	0.4906(5)	0.4113(6)	0.3137(6)
	SA2008	0.4906(13)	0.4113(21)	0.3137(30)
	SLHD	0.4906(9)	0.4113(9)	0.3137(9)
	LaPSO	0.4906(13)	0.4113(14)	0.3137(15)
	GA	0.4906(6)	0.4113(6)	0.3137(7)
	FastMm	0.4906(0)	0.4290(0)	0.3469(0)
5	SA	0.4907(5)	0.3351(6)	0.2715(7)
	SA2008	0.4907(13)	0.3451(17)	0.2715(26)
	SLHD	0.4907(9)	0.3351(9)	0.2715(11)
	LaPSO	0.4907(13)	0.3351(14)	0.2715(17)
	GA	0.4907(5)	0.3351(6)	0.2715(7)
	FastMm	0.4907(0)	0.4292(0)	0.2844(0)
6	SA	0.4821(8)	0.2974(9)	0.2421(10)
	SA2008	0.4821(18)	0.2974(25)	0.2414(33)
	SLHD	0.4821(12)	0.2974(13)	0.2414(14)
	LaPSO	0.4821(18)	0.2974(21)	0.2414(23)
	GA	0.4821(7)	0.2974(8)	0.2414(8)
	FastMm	0.4907(0)	0.3021(0)	0.2577(0)
7	SA	0.3961(11)	0.2758(12)	0.2237(13)
	SA2008	0.3961(23)	0.2811(30)	0.2181(38)
	SLHD	0.3961(16)	0.2758(17)	0.2197(18)
	LaPSO	0.3961(24)	0.2758(26)	0.2162(29)
	GA	0.3961(8)	0.2758(9)	0.2162(10)
	FastMm	0.4907(0)	0.3014(0)	0.2329(0)

	k	2	3	4
n		Min(CPU)	Min(CPU)	Min(CPU)
8	SA	0.3961(12)	0.2657(13)	0.2064(14)
	SA2008	0.3961(24)	0.2653(29)	0.2046(37)
	SLHD	0.3961(18)	0.2647(19)	0.2043(21)
	LaPSO	0.3961(26)	0.2556(29)	0.1907(32)
	GA	0.3961(8)	0.2556(9)	0.1907(10)
	FastMm	0.4907(0)	0.3501(0)	0.2049(0)
10	SA	0.3753(21)	0.2430(23)	0.1882(24)
	SA2008	0.3631(41)	0.2367(48)	0.1820(57)
	SLHD	0.3696(31)	0.2437(32)	0.1849(34)
	LaPSO	0.3631(46)	0.2257(49)	0.1736(53)
	GA	0.3631(14)	0.2270(14)	0.1732(15)
	FastMm	0.4816(0)	0.2758(0)	0.1844(0)
12	SA	0.3602(24)	0.2279(27)	0.1721(30)
	SA2008	0.3338(44)	0.2115(52)	0.1651(61)
	SLHD	0.3567(35)	0.2275(39)	0.1739(43)
	LaPSO	0.3338(51)	0.2044(57)	0.1524(64)
	GA	0.3338(14)	0.2024(16)	0.1534(18)
	FastMm	0.4696(0)	0.2475(0)	0.1608(0)
14	SA	0.3496(37)	0.2187(44)	0.1622(48)
	SA2008	0.3313(66)	0.2030(82)	0.1544(92)
	SLHD	0.3478(54)	0.2145(63)	0.1600(68)
	LaPSO	0.3256(77)	0.1865(92)	0.1407(99)
	GA	0.3189(21)	0.1872(25)	0.1407(27)
	FastMm	0.4908(0)	0.2259(0)	0.1677(0)

Table 2.5 shows the minimum ϕ_p value (Morris & Mitchell, 1995) under the L_1 distance (i.e., Manhattan distance) and the average CPU time for each case. Note that the designs from the FastMm method are proved to be optimal when their sizes are $n \times (n-1)$ with n being any odd prime and $n \times n$ with $2n+1$ or $n+1$ being prime (L. Wang et al., 2018). Thus, all designs by the algebraic construction FastMm have the smallest ϕ_p values in Table 2.5 and they can be found instantly. Among the five search algorithms, the GA has the smallest ϕ_p values and the lowest average CPU time. It can also find the same optimal designs by the FastMm method in some cases. The LaPSO performs similarly to the GA in terms of the ϕ_p values, but it requires much longer CPU time. Clearly, for such special design sizes mentioned in L. Wang et al., 2018, it is recommended to use the algebraic construction FastMm. If a search algorithm will be used, the GA is recommended.

Since the L_2 distance (i.e. Euclidean distance) is the most poplar in practice, we will focus on the maximin L_2 distance designs from now on. In Table 2.6, we show the results for designs with run sizes between 4 and 14. It is seen that when $k = 2$ with $n = 4$ or $n = 5$, all methods can give the same

Table 2.7: Minimum ϕ_p and Average CPU Time (in minutes) with the Rule of Thumb Sizes under the L_2 Distances.

$n \times k$	20×2	30×3	40×4	50×5	60×6	70×7	80×8
	Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)
SA	0.3344(1.4)	0.1707(3.3)	0.0999(7.9)	0.0678(13.7)	0.0499(22.0)	0.0380(37.6)	0.0292(55.4)
SA2008	0.3021(2.4)	0.1471(5.3)	0.0881(12.3)	0.0589(21.2)	0.0428(33.1)	0.0332(36.9)	0.0266(79.6)
SLHD	0.3070(2.1)	0.1643(4.8)	0.1002(11.2)	0.0685(19.6)	0.0490(31.0)	0.0373(43.2)	0.0297(76.3)
LaPSO	0.2849(3.0)	0.1283(6.8)	0.0753(16.0)	0.0512(27.8)	0.0378(44.9)	0.0292(75.9)	0.0236(99.4)
GA	0.2830(0.8)	0.1252(1.7)	0.0740(4.0)	0.0499(7.0)	0.0371(11.1)	0.0287(18.9)	0.0232(28.0)
FastMm	0.4905(0)	0.1565(0)	0.0895(0)	0.0634(0)	0.0450(0)	0.0342(0)	0.0280(0)

minimum ϕ_p, and the FastMm has the lowest CPU time. When $k = 2$ with $n = 6, 7, 8$, all the search algorithms give the same minimum ϕ_p, and the GA has the lowest CPU time. When $k = 2$ with $n = 10$ or $n = 12$, the SA2008, LaPSO, and GA give the same minimum ϕ_p. For the size 14×2, the GA gives the smallest minimum ϕ_p. When $k = 3$ with $n \leq 7$ and $k = 4$ with $n \leq 6$, almost all the search algorithms give the same minimum ϕ_p, and the GA has the lowest CPU time. For all other cases, it is seen that the GA and LaPSO have similar minimum ϕ_p and they outperform other search algorithms, but GA requires the minimum computing time.

In Table 2.7, we consider designs with the rule of thumb run size ($n = 10k$) for computer experiments (Chapman et al., 1994; Harari et al., 2018; Jones et al., 1998; Loeppky et al., 2009). From Table 2.7, it is seen that when both n and k increase, the GA tends to require the lowest CPU time among the five search algorithms. In most cases, the GA and LaPSO algorithms provide better results than others in terms of the ϕ_p values, where the former often outperforms the latter. The LaPSO has the highest CPU time in general. We would recommend the GA when the computational resources are limited.

We further display the boxplots for the ϕ_p values of some designs with the rule of thumb run sizes in Figure 2.1. For the case of $n = 20$ and $k = 2$ (top left panel in Figure 2.1), the GA gives the smallest median ϕ_p and the LaPSO is less stable compared to other algorithms. For the other three panels, the GA again outperforms all others. When the design sizes become larger, the advantage of using the GA tends to be more obvious.

In Figure 2.2, we illustrate the convergence of each algorithm for different design sizes. For the case of $n = 20$ and $k = 2$ (top left panel in Figure 2.2), the SA converges around the 1300th iteration, while the rest algorithms converge before the 500th iteration. For the cases of 40×4 and 60×6 (top right and bottom left panels in Figure 2.2), the SLHD has an improvement before the 2000th iteration while the rest algorithms stopped around the 1200th iteration. The GA converges faster than all others with good performances, and the LaPSO is the second best. Note that the GA exchanges entire columns for generating new candidate designs, while other methods exchange elements. Element-wise operations work well when n and k are relatively small. Yet, when the design sizes become larger, the numbers of elements increase exponentially and the strategy of exchanging elements may become less efficient.

Figure 2.1: Boxplots of ϕ_p Values from Different Algorithms with the Rule of Thumb Sizes.

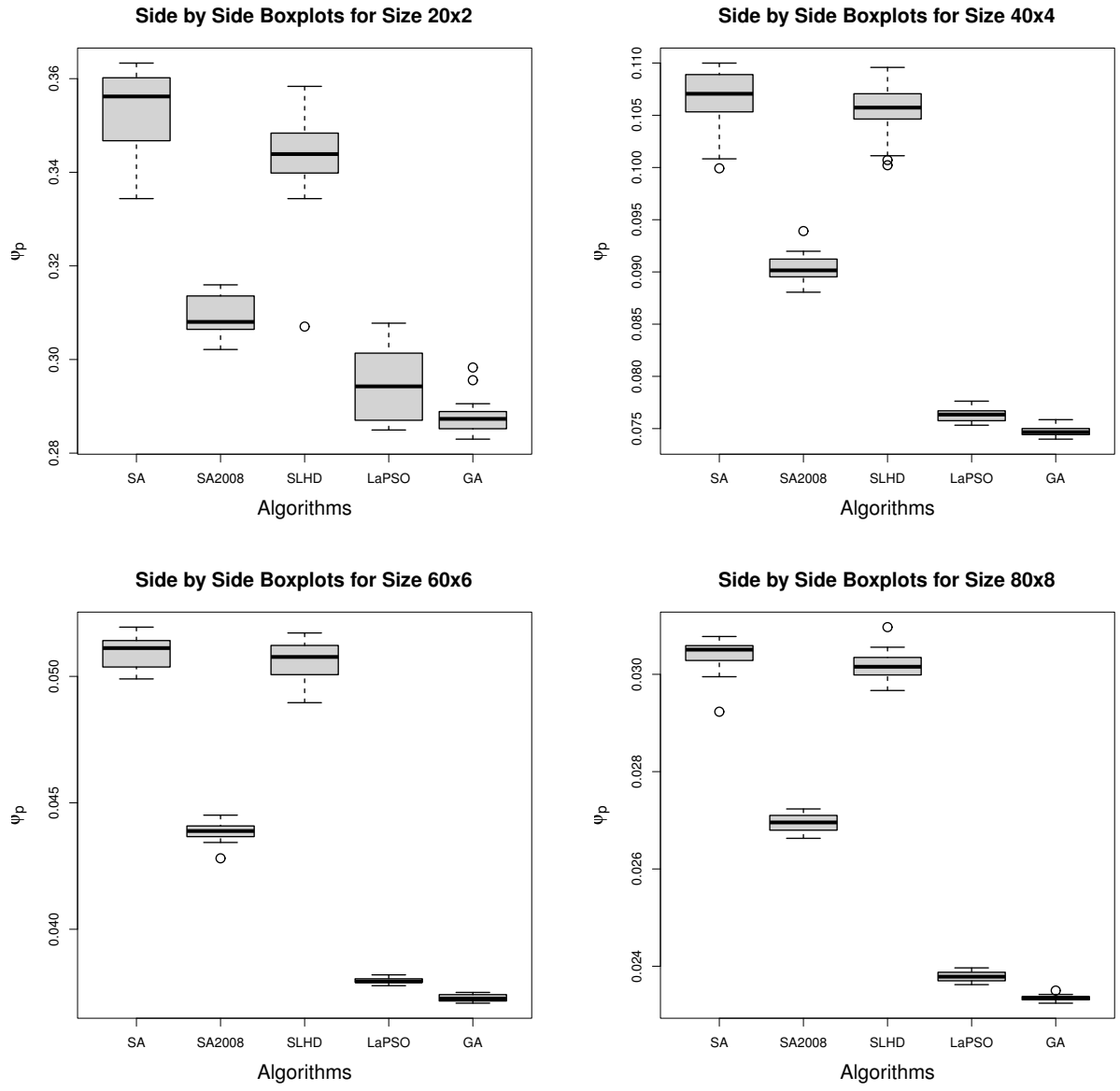


Figure 2.2: Convergence Curves for Different Algorithms with the Rule of Thumb Sizes.

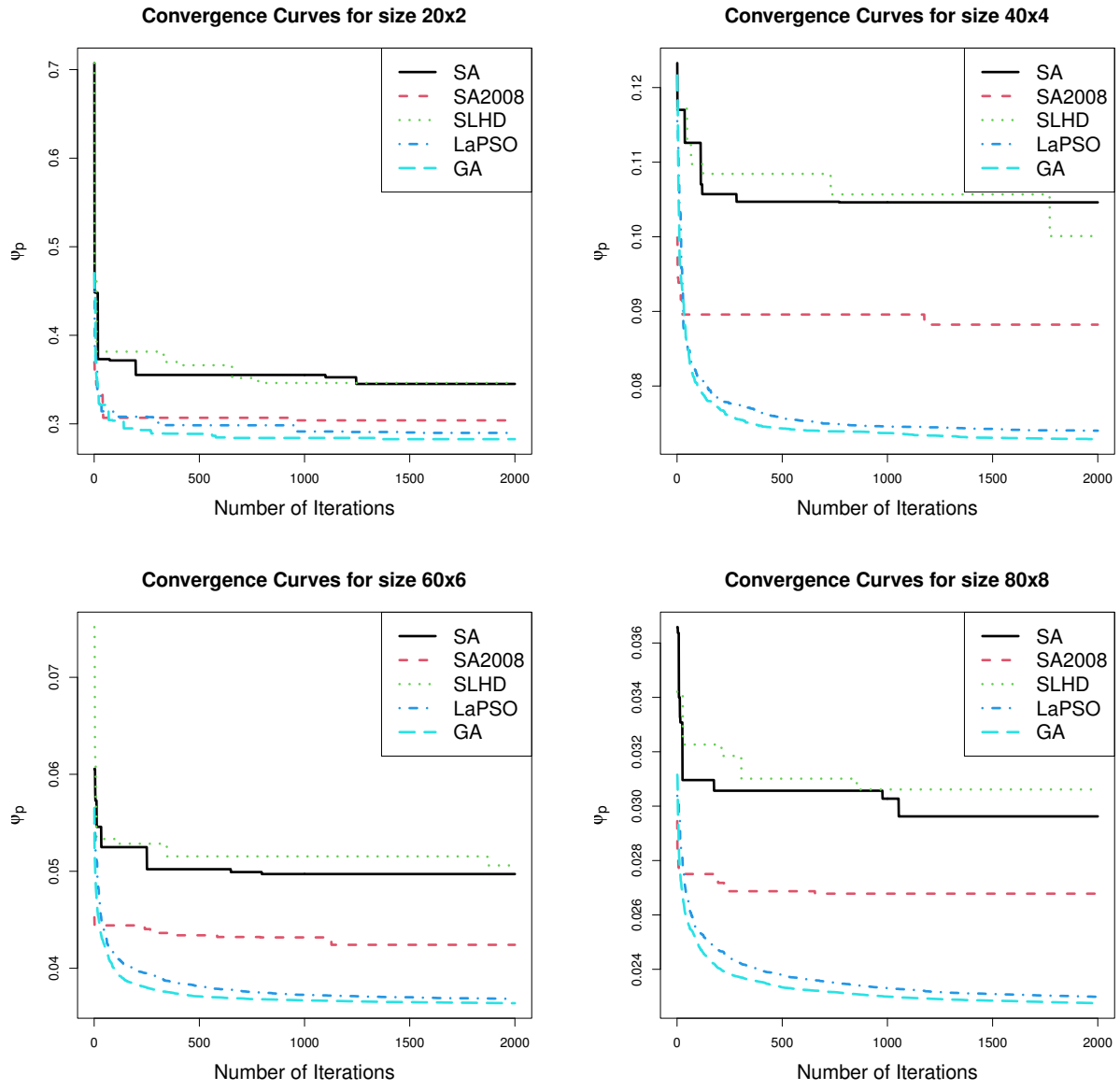


Table 2.8: Minimum ψ and Average CPU Time (in seconds) with Run Size Between 4 and 14.

	k	2	3	4
n		Min(CPU)	Min(CPU)	Min(CPU)
4	MaxPro	0.4513(0)	0.4705(0)	0.4454(0)
	LaPSO	0.4513(5)	0.4705(5)	0.4454(6)
	GA	0.4513(3)	0.4705(3)	0.4454(4)
5	MaxPro	0.3771(0)	0.3561(0)	0.3382(0)
	LaPSO	0.3771(5)	0.3561(6)	0.3382(7)
	GA	0.3771(3)	0.3561(4)	0.3382(4)
6	MaxPro	0.3154(0)	0.2633(0)	0.2724(0)
	LaPSO	0.3154(6)	0.2633(6)	0.2551(7)
	GA	0.3154(3)	0.2633(4)	0.2551(4)
7	MaxPro	0.2511(0)	0.2300(0)	0.2301(0)
	LaPSO	0.2511(6)	0.2184(7)	0.2113(8)
	GA	0.2511(3)	0.2184(4)	0.2113(5)

	k	2	3	4
n		Min(CPU)	Min(CPU)	Min(CPU)
8	MaxPro	0.2240(0)	0.2072(0)	0.2021(0)
	LaPSO	0.2240(8)	0.1737(9)	0.1763(11)
	GA	0.2240(4)	0.1737(4)	0.1763(5)
10	MaxPro	0.1800(0)	0.1682(0)	0.1625(0)
	LaPSO	0.1685(9)	0.1404(11)	0.1307(13)
	GA	0.1685(4)	0.1386(5)	0.1304(5)
12	MaxPro	0.1461(0)	0.1371(0)	0.1322(0)
	LaPSO	0.1330(9)	0.1115(11)	0.1010(13)
	GA	0.1330(4)	0.1120(4)	0.0990(5)
14	MaxPro	0.1299(0)	0.1174(0)	0.1077(0)
	LaPSO	0.1149(15)	0.0926(18)	0.0826(20)
	GA	0.1145(6)	0.0914(6)	0.0819(7)

Table 2.9: Minimum ψ and Average CPU Time (in minutes) with the Rule of Thumb Sizes.

$n \times k$	20×2	30×3	40×4	50×5	60×6	70×7	80×8
	Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)
MaxPro	0.0870(0)	0.0510(0)	0.0335(0)	0.0227(0)	0.0163(0)	0.0132(0)	0.0101(0)
LaPSO	0.0751(0.4)	0.0343(0.8)	0.0204(2.2)	0.0138(3.7)	0.0098(4.4)	0.0075(6.7)	0.0059(9.4)
GA	0.0749(0.1)	0.0335(0.2)	0.0193(0.5)	0.0128(0.9)	0.0093(1.1)	0.0071(1.6)	0.0056(2.3)

2.5.2 Results on Maximum Projection LHDs

Joseph et al., 2015 adopted an SA algorithm for identifying MaxPro LHDs, which is implemented by the R package `MaxPro` (Ba & Joseph, 2018) (denoted as "MaxPro" in the tables). Here, we compare it with the LaPSO and GA algorithms implemented by our developed R package `LHD`. The target is to find MaxPro LHDs via minimizing the objective function ψ defined in Equation (2.2). For each design size, we run every algorithm 20 times and show the best results (i.e. the minimum ψ values) along with the average CPU time in Tables 2.8 and 2.9.

In Table 2.8, when the design sizes are small (e.g. $k = 2$ with $n \leq 8$, $k = 3$ with $n \leq 6$ and $k = 4$ with $n \leq 5$), all the three algorithms give the same minimum ψ values, and the MaxPro has the lowest CPU time. For the rest design sizes (except for the case of 12×3), it is seen that the GA will lead to the best results. The LaPSO also performs better than the MaxPro method and its ψ values are close to those of GA. In Table 2.9, we show additional results for designs with the rule of thumb run sizes. It is seen that for relatively large design sizes, the GA finds the best designs with the smallest ψ values. The MaxPro requires the lowest CPU time. The LaPSO generally outperforms the MaxPro but requires the highest CPU time among the three.

Figure 2.3: Boxplots of ψ values from Different Algorithms with the Rule of Thumb Design Sizes.

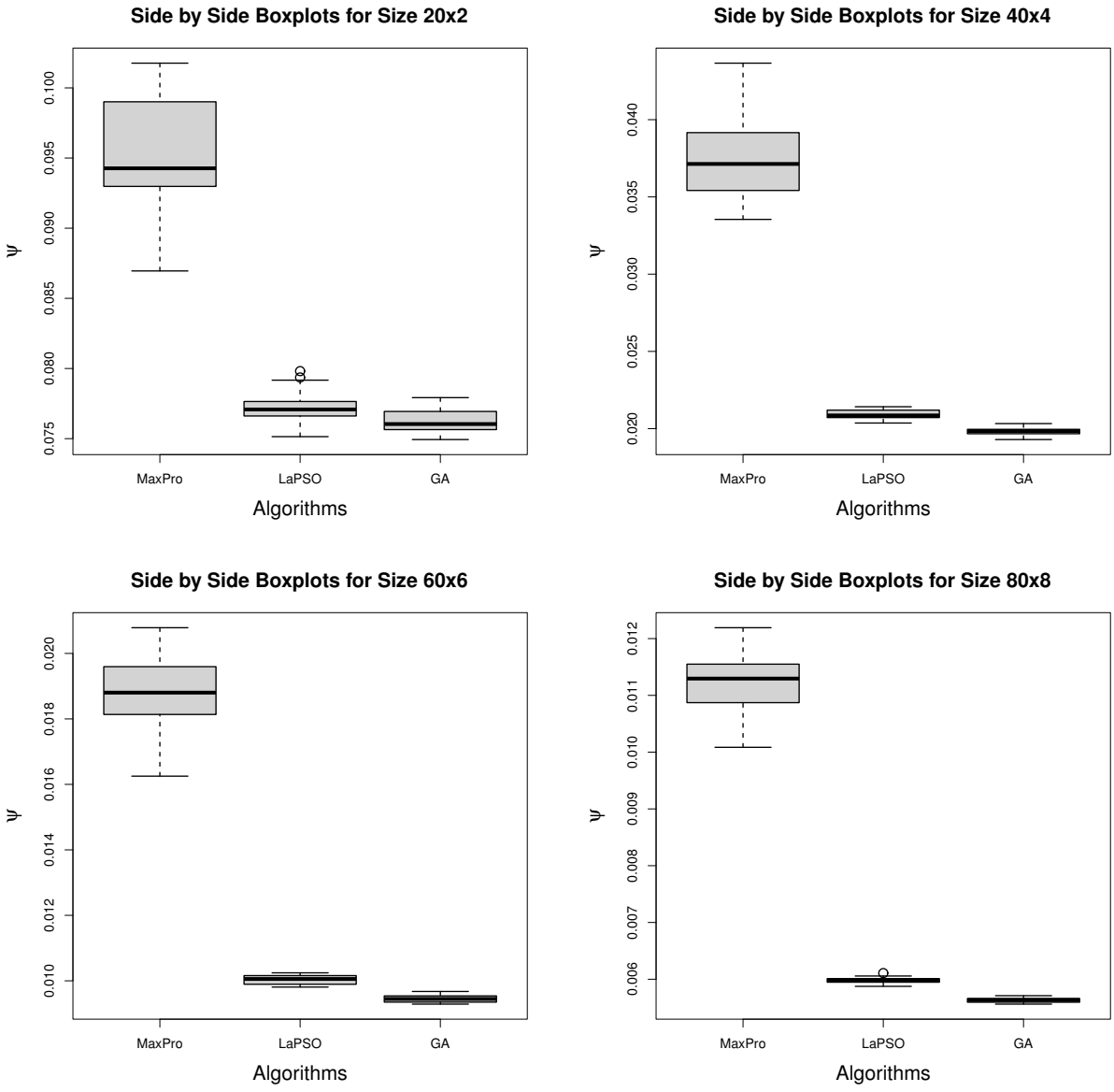


Figure 2.3 illustrates the boxplots of ψ values for some cases with the rule of thumb design sizes. From the top left panel in Figure 2.3 ($k = 2$ and $n = 20$), it is seen that the MaxPro method is less stable compared to others, where its boxplot is more stretched. The GA identifies the best design and is the most stable. From the other three panels in Figure 2.3, the GA again outperforms other methods. As design size increases, the advantage of the GA tends to be more obvious. Generally speaking, the proposed GA is recommended for constructing Maxpro LHDs with moderate and large sizes. The CPU time for the GA approach is often reasonable; while, the MaxPro is the fastest.

2.5.3 Results on Orthogonal and Nearly Orthogonal LHDs

In this part, we aim to identify orthogonal LHDs (OLHDs) and nearly orthogonal LHDs (NOLHDs) via minimizing the ave($|q|$) or max$|q|$ criteria defined in Equation (2.3). In Tables 2.10 to 2.12, we show the results for some designs with run sizes under 2^8. In these tables, "CM" represents whether there is an available algebraic construction method for the given design size. If "CM" is "Yes", we should follow the algebraic method to get the corresponding OLHD; otherwise, search algorithms are needed. Please refer to Table 2.3 in Section 2.4.2 for a summary on the available algebraic constructions for different design sizes.

In Table 2.10, algebraic constructions exist for two cases (8×4 and 9×4). The SA2008 can also identify these two OLHDs, where both the ave($|q|$) and max$|q|$ are 0. For the rest cases, the SA2008 is capable of identifying NOLHDs whose ave($|q|$) and max$|q|$ are very small. Generally speaking, the SA2008 outperforms all other search algorithms except for the case of 10×6 where the GA performs better. In Table 2.11, we further compare the SA2008 and the GA for some larger design sizes. Note that the SA, the SLHD and the LaPSO perform clearly worse than the SA2008 and the GA in our study, and we do not include them in Tables 2.11 to 2.12 for conciseness. When $n \geq 16$ and $k \geq 6$, it is seen that the GA provides the best results whose ave($|q|$) and max$|q|$ are very small. Its CPU time is also much lower than that of the SA2008.

In Table 2.12, we show additional cases with even larger run sizes. It is seen that there are very few cases where algebraic constructions are available. From Table 2.12, we can see that the GA gives the best designs with max$|q|$ less than 0.025 for all cases using reasonable CPU time. Moreover, in Table 2.13, we further consider some cases with the rule of thumb design sizes, where all five algorithms available in our developed R package `LHD` are compared. For the case of 20×2, all the five algorithms successfully identified the OLHDs, where the GA and the SA have the lowest CPU time. For the cases of $k \geq 3$, the GA gives the smallest max$|q|$ values (less than 0.0024) and requires the lowest CPU time for all cases. Overall speaking, the SA2008 has the best performance for small designs (e.g. $n < 16$) and the GA should be used for moderate and large designs.

Table 2.10: Minimum ave($|q|$), max$|q|$, and Average CPU Time (in seconds) with Run Size Below 2^4.

$n \times k$	Criteria	SA	SA2008	SLHD	LaPSO	GA	CM
7×4	ave($\|q\|$)	0.0298	0.0060	0.0238	0.0060	0.0179	No
	max$\|q\|$	0.0714	0.0357	0.0714	0.0357	0.0357	
	CPU	22	47	30	47	14	
8×4	ave($\|q\|$)	0.0198	0.0000	0.0317	0.0079	0.0079	Yes
	max$\|q\|$	0.0476	0.0000	0.0476	0.0238	0.0238	
	CPU	14	34	20	32	10	
9×4	ave($\|q\|$)	0.0278	0.0000	0.0194	0.0083	0.0000	Yes
	max$\|q\|$	0.0500	0.0000	0.0500	0.0167	0.0000	
	CPU	14	36	21	32	10	
10×4	ave($\|q\|$)	0.0283	0.0061	0.0283	0.0061	0.0061	No
	max$\|q\|$	0.0545	0.0061	0.0545	0.0061	0.0061	
	CPU	22	56	32	48	14	
10×6	ave($\|q\|$)	0.0877	0.0174	0.1030	0.0246	0.0166	No
	max$\|q\|$	0.2121	0.0424	0.2121	0.0545	0.0303	
	CPU	64	135	88	136	37	
12×6	ave($\|q\|$)	0.0932	0.0121	0.0769	0.0154	0.0145	No
	max$\|q\|$	0.1748	0.0350	0.1818	0.0280	0.0280	
	CPU	63	144	89	137	37	
14×6	ave($\|q\|$)	0.0769	0.0086	0.0769	0.0125	0.0092	No
	max$\|q\|$	0.1473	0.0198	0.1516	0.0242	0.0198	
	CPU	44	116	65	96	26	

Table 2.11: Minimum ave($|q|$), max$|q|$, and Average CPU Time (in seconds) with Run Size Between 2^4 and $2^5 - 1$.

$n \times k$	Criteria	SA2008	GA	CM
16×6	ave($\|q\|$)	0.0098	0.0076	No
	max$\|q\|$	0.0206	0.0118	
	CPU	120	25	
17×6	ave($\|q\|$)	0.0101	0.0067	Yes
	max$\|q\|$	0.0221	0.0147	
	CPU	137	26	
18×6	ave($\|q\|$)	0.0118	0.0072	No
	max$\|q\|$	0.0258	0.0114	
	CPU	123	23	
17×8	ave($\|q\|$)	0.0207	0.0179	Yes
	max$\|q\|$	0.0613	0.0343	
	CPU	189	43	

$n \times k$	Criteria	SA2008	GA	CM
18×8	ave($\|q\|$)	0.0241	0.0197	No
	max$\|q\|$	0.0526	0.0361	
	CPU	195	42	
22×8	ave($\|q\|$)	0.0211	0.0099	No
	max$\|q\|$	0.0536	0.0220	
	CPU	231	43	
28×8	ave($\|q\|$)	0.0155	0.0070	No
	max$\|q\|$	0.0383	0.0142	
	CPU	274	40	
30×8	ave($\|q\|$)	0.0150	0.0070	No
	max$\|q\|$	0.0389	0.0131	
	CPU	293	42	

Table 2.12: Minimum ave($|q|$), max$|q|$, and Average CPU Time (in minutes) with Run Size Between 2^5 and 2^8.

$n \times k$	Criteria	GA	CM
33×8	ave($|q|$)	0.0058	Yes
	max$|q|$	0.0124	
	CPU	1.1	
34×8	ave($|q|$)	0.0048	No
	max$|q|$	0.0096	
	CPU	1.1	
48×10	ave($|q|$)	0.0094	No
	max$|q|$	0.0156	
	CPU	1.7	
64×10	ave($|q|$)	0.0063	No
	max$|q|$	0.0114	
	CPU	1.7	
65×10	ave($|q|$)	0.0038	Yes
	max$|q|$	0.0087	
	CPU	1.7	

$n \times k$	Criteria	GA	CM
68×10	ave($|q|$)	0.0048	No
	max$|q|$	0.0088	
	CPU	1.7	
96×12	ave($|q|$)	0.0096	No
	max$|q|$	0.0178	
	CPU	1.8	
128×14	ave($|q|$)	0.0136	No
	max$|q|$	0.0241	
	CPU	3.0	
192×14	ave($|q|$)	0.0089	No
	max$|q|$	0.0170	
	CPU	3.9	
256×16	ave($|q|$)	0.0119	No
	max$|q|$	0.0216	
	CPU	4.5	

Table 2.13: Minimum ave($|q|$), max$|q|$, and Average CPU Time (in minutes) with the Rule of Thumb Sizes.

	$n \times k$	20×2	30×3	40×4	50×5	60×6	70×7	80×8
Criteria		Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)	Min(CPU)
ave($|q|$)	SA	0(0.1)	0.0013(0.1)	0.0126(0.2)	0.0281(0.8)	0.0304(1.0)	0.0401(1.4)	0.0398(2.1)
	SA2008	0(1.1)	0.0002(2.0)	0.0008(3.8)	0.0024(10.0)	0.0054(14.4)	0.0066(20.7)	0.0091(30.3)
	SLHD	0(0.2)	0.0022(0.4)	0.0091(0.8)	0.0266(2.3)	0.0317(3.4)	0.0399(4.8)	0.0477(7.0)
	LaPSO	0(0.2)	0.0002(0.4)	0.0004(0.6)	0.0006(1.9)	0.0006(2.6)	0.0011(3.6)	0.0020(5.0)
	GA	0(0.1)	0.0002(0.2)	0.0004(0.2)	0.0004(0.5)	0.0006(0.6)	0.0011(0.8)	0.0011(1.1)
max$|q|$	SA	0(0.1)	0.0020(0.1)	0.0218(0.2)	0.0514(0.8)	0.0702(1.0)	0.1045(1.4)	0.1000(2.1)
	SA2008	0(1.1)	0.0002(2.0)	0.0015(3.8)	0.0053(10.0)	0.0103(14.4)	0.0204(20.7)	0.0259(30.3)
	SLHD	0(0.2)	0.0038(0.4)	0.0165(0.8)	0.0477(2.3)	0.0704(3.4)	0.0891(4.8)	0.0996(7.0)
	LaPSO	0(0.2)	0.0002(0.4)	0.0008(0.6)	0.0009(1.9)	0.0013(2.6)	0.0022(3.6)	0.0040(5.0)
	GA	0(0.1)	0.0002(0.1)	0.0008(0.2)	0.0008(0.5)	0.0011(0.6)	0.0022(0.8)	0.0024(1.1)

Table 2.14: Recommended Algorithm (or Method) for Maximin Distance LHDs.

n	$k = 2$	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$	$k = 8$	$k = 9$	$k = 10$
4 to 5	FastMm	GA	GA						
6 to 7	GA	GA	GA						
8 to 15	GA	GA	GA	GA	GA				
16 to 31	GA	GA	GA	GA	GA	GA	GA		
32 to 63	GA	GA	GA	GA	GA	GA	GA	GA	GA

2.6 Conclusion and Recommendation

In this chapter, we target on constructing three types of commonly used optimal LHDs: the maximin distance LHDs, the maximum projection LHDs and the (nearly) orthogonal LHDs. We have summarized, improved and compared some currently popular search algorithms, including the SA (Morris & Mitchell, 1995), SA2008 (Joseph & Hung, 2008), SLHD (Ba et al., 2015), LaPSO (R.-B. Chen et al., 2013) and GA (Liefvendahl & Stocki, 2006), along with some widely used algebraic constructions (Butler, 2001; Cioppa & Lucas, 2007; C. D. Lin et al., 2009; Sun et al., 2010; Tang, 1993; L. Wang et al., 2018; Ye, 1998). We developed efficient implementations of these methods and integrate them into the R package `LHD`. We aim to provide guidance and an easy-to-use tool for practitioners to find appropriate experimental designs. Algebraic constructions are preferred when available, especially for large designs. Search algorithms are used to generate optimal LHDs with flexible sizes.

From the numerical studies in Section 2.5, we can see that the GA is the most reliable approach to generate optimal LHDs, which provides a good balance between the performance and the computing time. It outperforms other search algorithms for moderate and large design sizes. The LaPSO performs well but often requires much longer computing time. For generating MaxPro LHDs, the R package `MaxPro` (Ba & Joseph, 2018) is fast, but its performance is often worse than that of the GA (implemented by the R package `LHD`). We provide two tables summarizing the recommended methods for different design sizes in Tables 2.14 to 2.16.

The search algorithms discussed in this chapter can also generate other types of efficient experimental designs. An interesting future research is to expand the scope of this chapter and consider other design types including the fractional factorial designs (Dean et al., 2017), supersaturate designs (D. K. J. Lin, 1993) and order-of-addition designs (Peng et al., 2019). We will develop a more comprehensive R package implemented with efficient coding for practitioners.

Table 2.15: Recommended Algorithm for Maximum Projection LHDs.

n	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$
4 to 5	MaxPro	MaxPro	MaxPro						
6	MaxPro	MaxPro	GA						
7	MaxPro	GA	GA						
8	MaxPro	GA	GA	GA	GA				
9 to 15	GA	GA	GA	GA	GA				
16 to 31	GA	GA	GA	GA	GA	GA	GA		
32 to 63	GA	GA	GA	GA	GA	GA	GA	GA	GA

Table 2.16: Recommended Algorithm (or Method) for Orthogonal and Nearly Orthogonal LHDs

n	$k=2$	$k=3$	$k=4$
4	Sun10	GA	GA
5	Sun10/Ye98	GA	GA
6	GA	GA	GA
7	GA	GA	SA2008
8	Sun10	GA	Sun10
9	Sun10	GA	Sun10/Ye98
10	GA	GA	GA
11	GA	SA2008	SA2008
12	Sun10	SA2008	SA2008
13	Sun10	GA	SA2008
14	GA	SA2008	GA
15	GA	GA	GA
16	Sun10	GA	Sun10
17	Sun10	GA	Sun10
18	GA	GA	GA
19	GA	GA	GA
$4r$ or $4r+1$	Sun10		
$4r+2$ or $4r+3$	GA		
$8r$ or $8r+1$			Sun10

2.7 Research Outcomes

This research let me understand different types of LHDs and construction theories for them. I coded all the construction methods and search algorithms introduced in this chapter via R, and I published an R library `LHD`. This package have almost $20,000$ number of downloads, which can be a good contribution to the design community. I gained extensive experience in R coding as well as how to publish an R package. More importantly, when I was coding all the algorithms, I became familiar with the strength and weakness for each algorithm. This inspires me to create a brand new algorithm that could have better performance than other algorithms in current literature.

Chapter 3

Lioness Algorithm for Finding Optimal Design of Experiments

3.1 Introduction

Design of experiments (DOE) is an important field in statistics, which studies how to systematically conduct, analyze and interpret experimentation to investigate the input-output relationship (Dean et al., 2017). A thorough DOE may minimize the experimental error and gain the most information when limited amount of data are being collected, which is particularly important if resources are limited. DOE are commonly used in scientific researches and industrial productions, and a wide range of applications are covered (Wu & Hamada, 2021). As motivating examples, consider an optimal design of continuous inputs is used for metal cutting in the manufacturing process (Antony, 2014), which aims to explore the relationship between the depth of cut, the cutting speed and the feed rate with the surface finish of the manufactured part. Additionally, in clinical trials, researchers recently found that the orders-of-addition of five FDA approved chemotherapeutics with fixed doses may have significant impact on the overall treatment efficacy for lymphoma treatment (A. Wang et al., 2020; Xiao & Xu, 2021) and optimal order-of-addition design for sequence inputs are needed (D. K. Lin & Peng, 2019). In addition to physical experiments, computer experiments that simulate physical phenomenon become increasingly popular in recent decades, which requires space-filling Latin Hypercube designs (LHD) (Gramacy, 2020; Mak et al., 2018; Pukelsheim, 2006). For example, Wu and Hamada, 2021 adopted a space-filling LHD to collect data from computer models for finding efficient airfoil setting with large gliding ratio. Algebraic constructions based on theoretical results for constructing such designs only work for some special cases and search algorithms are needed to identify optimal designs with flexible sizes.

With a significant improvement of computing power in the recent years, a variety of meta-heuristics optimization algorithms have been proposed to solve optimization problems, such as minimizing the cost and wastage in manufacturing, minimizing the pollution discharge in production, maximizing outcomes in financial portfolio and maximizing the performance and the efficiency of engine (X.-S. Yang, 2020). Such problems often can be formulated by complex models or black-box functions whose solutions are

nontrivial. Meta-heuristic optimization algorithms are often used for optimizing such functions due to their superior performances (Mirjalili et al., 2014).

Single solution based algorithms and population based algorithms are used in practice. The former begins with one random solution and iteratively improves it; standard methods typically include the iterated local search (Lourenço et al., 2019), simulated annealing (SA, Kirkpatrick et al., 1983) and variable neighborhood search (Hansen et al., 2010) techniques. On the other hand, population-based algorithms start with multiple random solutions (denoted as a population), and the entire population would be improved over iterations. Popular population-based algorithms include the ant colony optimization (Dorigo et al., 2006), genetic algorithms (GAs, J. Holland, 1975) and particle swarm optimization (PSO, Kennedy and Eberhart, 1995).

Another way to classify meta-heuristics algorithms is the evolutionary algorithms versus swarm intelligence algorithms (Mirjalili et al., 2014; Stokes et al., 2020; X.-S. Yang, 2020). Evolutionary algorithms are inspired by biological evolution. Such algorithms involve a set of random solutions and iteratively select the top individuals to form new generations as the future generation is more likely to have better performance than the previous generations. The genetic algorithm (J. Holland, 1975) may be the most popular evolutionary algorithm, which has been widely studied and implemented for many applications (C. D. Lin et al., 2015). Other popular evolutionary algorithms include the differential evolution (Storn & Price, 1997), the evolutionary programming (Yao et al., 1999) and the biogeography-based optimization (Simon, 2008). Swarm intelligence (SI) algorithms are usually inspired and named after the natural processes or social behaviors of animals such as swarms, bird flocks and herds. SI algorithms often have a collection of agents that search and move within the search space in a manner of mimicking the collective and social intelligence of animals. There is a rich literature on SI algorithms and popular ones include the ant colony optimization (Dorigo et al., 2006), the artificial bee colony (Basturk, 2006), the bat algorithm (X.-S. Yang, 2010), the cuckoo search (X.-S. Yang & Deb, 2009), the firefly algorithm (X.-S. Yang, 2009), the grey wolf optimization (Mirjalili et al., 2014), the jumping frogs optimization (Garcia & Pérez, 2008) and the particle swarm optimization (PSO, Kennedy and Eberhart, 1995. Those SI algorithms are often global optimum oriented and are efficient in many different optimization problems (Kennedy, 2006; X.-S. Yang, 2020).

In DOE, optimal designs can be identified using meta-heuristics algorithms via optimizing certain optimality criteria. Model-specific optimal designs are popular in practice, where the optimality criteria are often related to information matrix of the corresponding model (Dean et al., 2017; Heredia-Langner et al., 2003). For example, D-optimality criterion is to maximize the determinant of a model's information matrix, A-optimality criterion is to minimize the trace of the inverse of the information matrix, and E-optimality criterion is to maximize the minimum eigenvalue of the information matrix. Model free designs are also popular in practice as the model intended to be used may not be known before actually conducting the experiments (Fang et al., 2005; C. D. Lin et al., 2009). In computer experiments, space-filling Latin hypercube designs (LHDs, McKay et al., 1979) may be the most popular designs (Gramacy, 2020) due to their deterministic characteristics. Orthogonal designs are also popular in both physical and computer

experiments as all pairwise correlations between factors are zero (Butler, 2001; C. D. Lin et al., 2009; Sun et al., 2010).

Specifically, we consider the types of space-filling LHDs widely used in computer experiments where different optimality criteria are used. Johnson et al., 1990 proposed the maximin distance criterion to achieve the overall space-filling in the entire domain, Joseph et al., 2015 proposed the maximum projection criterion considering the space-filling for all possible projections, and Georgiou, 2009 proposed the average absolute correlation criterion and the maximum absolute correlation criterion to explore the orthogonality of LHDs; see Section 3.2 for a thorough review. Meta-heuristic optimization algorithms can be used for identifying optimal designs via optimizing these criteria (R.-B. Chen et al., 2013; Joseph et al., 2015; Joseph & Hung, 2008; Liefvendahl & Stocki, 2006; Lukemire et al., 2019; Morris & Mitchell, 1995; Stokes et al., 2020).

In this chapter, we propose a novel population-based swarm intelligence algorithm that is inspired by the hunting behaviors of lionesses, named as the lioness algorithm (LA). Compared to other swarm intelligence algorithms, the proposed LA has the following advantages and novelties. (1) The candidate solutions at each iteration in the LA are involved with the current global best as well as the second and third global bests, whereas most current works only consider the current global best. Thus, the LA is expected to have a more robust performance. (2) The LA requires less memory usage and CPU time, as its solutions at each iteration are updated through column exchanges, which is a different procedure compared to most current methods. (3) The LA is capable of identifying designs with continuous input variables as well as designs with discrete input variables. Note that popular algorithms including the SA (Kirkpatrick et al., 1983), the PSO (Kennedy & Eberhart, 1995), the ant colony optimization (Dorigo et al., 2006), the LaPSO (R.-B. Chen et al., 2013) and the GA (Liefvendahl & Stocki, 2006) can only work to either continuous or discrete designs. (4) LA can embed flexible mechanisms to avoid local optimum, which helps to deal with different types of designs. (5) The LA starts with multiple random solutions to reduce the possibility of initializing with "bad" solutions at the first place. Specifically, the proposed LA is different from the grey wolf optimization (Mirjalili et al., 2014) in spite of the similarities in the names. The LA updates candidate solutions through column exchanges and embeds mechanisms to avoid local optimum, while the grey wolf optimization does not have such procedures.

To facilitate applications, we developed the R package `LA` available on the Comprehensive R Archive Network (https://cran.r-project.org/web/packages/LA/index.html). Practitioners with little or no background on design theory can easily find various types of optimal designs with the required run-sizes to satisfy their needs.

The rest of the chapter is organized as follows. Section 3.2 reviews various types of optimal designs. Section 3.3 demonstrates the proposed LA for constructing both the continuous and discrete optimal designs. Section 3.4 illustrates the superior performance of LA via numerical studies. Section 3.5 concludes and discusses with some potential future research.

3.2 Brief Review on Optimal Designs

This section reviews some popular design types and optimality criteria. A design that optimizes a specified optimality criterion is called an optimal design. We consider both continuous and discrete designs in this chapter. Factors in continuous designs can take any value within certain ranges, while factors in discrete designs contain only discrete elements (limited number of levels). Throughout this chapter we use $\mathbf{X}$ to denote a design matrix.

Discrete Designs

Latin hypercube designs (LHDs, McKay et al., 1979) are popular for computer experiments, since they have uniform one-dimensional projection property and thus avoid replications in each dimension (Santner et al., 2003). An LHD with n runs and k factors can be viewed as an n by k design matrix with each column being a random permutation of $1, \ldots, n$, where $n \geq 2$ and $k \geq 2$. A variety of optimality criteria have been proposed to measure the space-filling properties of LHDs, including the Maximin distance and the Minimax distance criteria (Johnson et al., 1990) and the uniformity discrepancies (Fang et al., 2005; Fang et al., 2002). In this chapter, we focus on the currently popular maximin distance criterion which seeks to scatter design points over the experimental domains such that the minimum distances between points are maximized.

Define the L_q-distance between two runs x_i and x_j of $\mathbf{X}$ as $d_q(x_i, x_j) = \left\{\sum_{m=1}^{k} |x_{im} - x_{jm}|^q\right\}^{1/q}$, where q is any integer. Two popular choices of q are $q = 1$ (Manhattan distance) and $q = 2$ (Euclidean distance). A design $\mathbf{X}$ is called the maximin L_q-distance design if it maximizes the minimum L_q-distance (i.e., it has the $\max\min d_q(x_i, x_j)$ for $1 \leq i < j \leq n$). To express the maximin distance criterion in a more convenient way, Morris and Mitchell, 1995 and Jin et al., 2005 proposed the ϕ_p criterion as

$$\phi_p = \left\{ \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_q(x_i, x_j)^{-p} \right\}^{1/p}. \tag{3.1}$$

The ϕ_p criterion in Equation (3.1) is asymptotically equivalent to the maximin distance criterion as $p \to \infty$, and $p = 15$ is usually sufficient in practice.

The maximin distance LHD focuses on the space-filling property in its full dimensional space, but its projection properties in all sub-dimensional spaces are not guaranteed. Joseph et al., 2015 proposed the maximum projection criterion that considers designs' space-filling properties in all possible dimensional spaces. A design $\mathbf{X}$ is called a maximum projection design if it minimizes the maximum projection criterion

$$\psi(\mathbf{X}) = \left\{ \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{\Pi_{l=1}^{k} (x_{il} - x_{jl})^2} \right\}^{1/k}. \tag{3.2}$$

From Equation (3.2), we can see that any two design points should be apart from each other in any projection in order to minimize the value of $\psi(\mathbf{X})$. Orthogonal (and nearly orthogonal) LHDs are another popular type of LHDs which aim to minimize the correlations among factors (Georgiou, 2009; Steinberg & Lin, 2006; Sun & Tang, 2017). Two major correlation-based criteria measuring designs' orthogonality are the average absolute correlation criterion and the maximum absolute correlation criterion (Georgiou, 2009), denoted as ave$(|q|)$ and max$|q|$, respectively. They are defined as:

$$\text{ave}(|q|) = \frac{2\sum_{i=1}^{k-1}\sum_{j=i+1}^{k}|q_{ij}|}{k(k-1)} \quad \text{and} \quad \max|q| = \max_{i,j}|q_{ij}|, \tag{3.3}$$

where q_{ij} is the correlation between the ith and jth columns in a design matrix $\mathbf{X}$. Orthogonal designs will have ave$(|q|) = 0$ and max$|q| = 0$, which may not be attainable for all design sizes. Designs with smaller ave$(|q|)$ or max$|q|$ are generally preferred in practice.

In the recent literature, order-of-addition designs (OofAs) has drawn much attention (Peng et al., 2019). OofAs are widely used in chemical and pharmaceutical studies (D. K. Lin & Peng, 2019; A. Wang et al., 2020). For example, in chemistry, when reactants are added into the apparatus in a sequential manner, the order of adding reactants can significantly affect the outcomes (D. K. Lin & Peng, 2019). To analyze such experiments, Van Nostrand, 1995 proposed the pairwise-order (PWO) model using an OofA. An OofA with n runs and k factors can be viewed as an n by k design matrix with each row being a distinct permutation of $1, \ldots, k$, where $k \geq 2$ and $2 \leq n \leq k!$ as there are at most $k!$ distinct orders. When $n = k!$, an OofA would be called a full OofA. Each OofA has its own PWO matrix with size $n \times \binom{k}{2}$. To generate a PWO matrix, the first step is to list all distinct pairs of elements in its OofA. For example, an OofA with $k = 4$ has the following pairs: $\{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}$. The elements in PWO matrix are generated according to the precedence situations for all pairs in each row of OofA. Let $\mathbf{Z}$ denote the PWO matrix of an OofA design $\mathbf{X}$, (i,j) denote a pair of elements where $1 \leq i < j \leq k$, and $\mathbf{x}_m$ denote the mth row of $\mathbf{X}$. For all the possible values of i and j, the mth row of $\mathbf{Z}$ is formed by the following logic:

$$\mathbf{z}_m = \begin{cases} +1, & \text{if} \quad i \text{ precedes } j \text{ in } \mathbf{x}_m\,, \\ -1, & \text{if} \quad j \text{ precedes } i \text{ in } \mathbf{x}_m\,. \end{cases}$$

For example, if the first row of $\mathbf{X}$ with $k = 4$ is given by $\{2,4,1,3\}$, then the first row of $\mathbf{Z}$ would be $\{-1,1,-1,1,1,-1\}$. With the PWO matrix $\mathbf{Z}$, the moment matrix can be defined as $\mathbf{M} = [\mathbf{1},\mathbf{Z}]^T[\mathbf{1},\mathbf{Z}]/n$, where $\mathbf{1}$ is a column vector containing n ones. A design $\mathbf{X}$ is called an optimal OofA if it optimizes $\mathbf{M}$ according to some optimality criteria. Popular criteria for optimizing $\mathbf{M}$ include maximizing the determinant of $\mathbf{M}$ (D-optimality criterion), maximizing the trace of the inverse of $\mathbf{M}$ (A-optimality criterion), maximizing the trace of the square of $\mathbf{M}$ (M.S.-optimality criterion), and maximizing minimum eigenvalue of $\mathbf{M}$ (E-optimality criterion). Note that LHDs and OofAs are different. Each column of an LHD is a random permutation of $1, 2, \ldots, n$, while each row of an OofA is a random permutation of $1, 2, \ldots, k$. Optimization for LHDs usually focuses on column-wise operations, while optimization for OofAs trends to focus on row-wise operations.

Continuous Designs

Elements in continuous design matrices can take any value within their domains. In this chapter, we focus on exact designs that may have more than one replicates on the support points. Optimal continuous designs are often model-specific. Minimizing the variances of the estimates of model parameters leads to the identification of optimal designs. For vector-valued parameters, the variance-covariance matrix of parameter estimates (i.e. the inverse of the model's information matrix) plays a key role in identifying optimal designs. Under a given model, the optimal design is identified by maximizing the model's information matrix according to certain optimality criterion.

In linear models, D-optimal design (denoted as $\mathbf{X}_D^*$) would be identified by maximizing the determinant of the information matrix $\mathbf{X}^T\mathbf{X}$ such that

$$\mathbf{X}_D^* = \underset{\mathbf{X}}{\operatorname{argmax}} |\mathbf{X}^T\mathbf{X}|.$$

For general cases, the global normalized D-optimality criterion is defined as

$$-\log \left| \sum_{i=1}^{k} p_i \big(\nabla f(x_i, \beta)\big) \big(\nabla f(x_i, \beta)\big)^T \right|,$$

where p_i is weight and $\nabla f(x_i, \beta)$ is the partial derivative of the mean function with respect to the model parameters β. When model errors are assumed to be independently and normally distributed, D-optimal designs minimize the volume of the joint confidence ellipsoid for model parameters, which lead to the most precise model parameters' estimators (Stokes et al., 2020).

Another popular optimality criterion is the A-optimality, which minimizes the trace of the inverse of information matrix. In linear models, the A-optimal design (denoted as $\mathbf{X}_A^*$) would be identified by minimizing the trace of $(\mathbf{X}^T\mathbf{X})^{-1}$ such that

$$\mathbf{X}_A^* = \underset{\mathbf{X}}{\operatorname{argmin}} \operatorname{tr}(\mathbf{X}^T\mathbf{X})^{-1}.$$

The A-optimality criterion minimizes the average variance of the model parameters' estimator. There are other popular optimality criteria such as E-optimality (maximizing the minimum eigenvalue of the information matrix) and G-optimality (minimizing the maximum prediction variance) etc. In Section 3.4, we briefly discuss the G-optimality criterion together with quadratic models.

Several optimization algorithms have been proposed in the literature to identify either optimal continuous designs or optimal discrete designs. Borkowski, 2003 implemented GA (J. Holland, 1975) to construct exact G-optimal designs. Liefvendahl and Stocki, 2006 modified the GA to search for space-filling LHDs. Qiu et al., 2014 applied PSO (Kennedy & Eberhart, 1995) to find optimal continuous designs for tumor growth models. R.-B. Chen et al., 2013 adapted the framework of PSO and proposed LaPSO for space-filling LHDs. Morris and Mitchell, 1995 customized SA algorithm (Kirkpatrick et al., 1983) to identify optimal space-filling LHDs. Following their work, Joseph and Hung, 2008 updated the

SA algorithm for both space-filling and orthogonal designs. Joseph et al., 2015 adapted the SA algorithm to search for maximum projection LHDs. Stokes et al., 2020 applied differential evolution algorithms (Price et al., 2006; Storn & Price, 1997) to find D-optimal designs for estimating the Arrhenius frequency parameter and activation temperature in the Arrhenius equation (Emich, 1900). Peng et al., 2019 proposed a bubble-sort exchange algorithm to search for D-optimal OofAs. Voelkel, 2019 proposed to use an exchange algorithm (Fedorov, 2013) to search for small-run efficient OofAs.

Usually, standard off-the-shelf algorithms are adapted to solve design problems, with adding equality or inequality constraints, as appropriate. One advantage of the proposed LA is that it does not require any constraint. Another advantage of the LA is its compatibility on both design types, whereas current popular optimization algorithms usually are not compatible for both. For example, popular optimization algorithms such as SA (Kirkpatrick et al., 1983) and PSO (Kennedy & Eberhart, 1995) can be used for optimizing continuous designs, but they cannot be directly implemented on discrete designs as calculations on discrete elements won't make sense. Adapted versions, such as adapted SA (Morris & Mitchell, 1995), LaPSO (R.-B. Chen et al., 2013) and adapted GA (Liefvendahl & Stocki, 2006), can be used for optimizing LHDs and OofAs; yet, they can not be used for continuous designs.

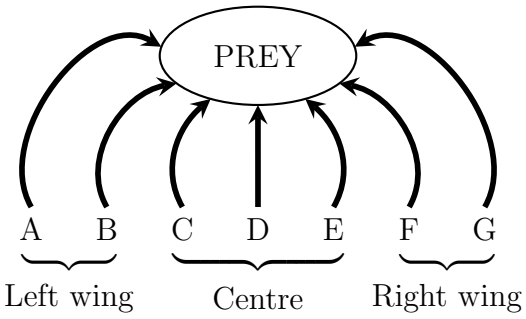
3.3 The Lioness Algorithm for Optimal Designs

3.3.1 Inspiration and General Framework

In a lion pride, male lions are responsible for protecting their territory, while lionesses take care of hunting. Lionesses usually hunt in a group. Stander, 1992 introduced three terminologies to describe the relative positions between lionesses and prey during their cooperative hunting, and they are called centre, left wing and right wing, which are shown in Figure 3.1. The centre lionesses (C, D, and E in Figure 3.1) are at the closest position toward to prey, and they charge prey at the first place. Left wing (A and B) and right wing (F and G) lionesses encircle prey and remain cautious to prevent the prey from running away. The entire hunting process involves iteratively approaching and encircling prey until they get captured. In this subsection, we first demonstrate the general framework of LA and summarize it in Algorithm 4 at the end. In the next subsection, we illustrate how this framework can be applied to find different optimal designs.

The four-step procedure of the lioness algorithm (LA) is inspired by the hunting process of lionesses. When lionesses start to look for prey, they assemble at the first place and then make the next move. The first step of LA mimics such behavior by considering the optimal design as the prey and generating random design candidates as lionesses assembling. When lionesses spot the prey, the relative positions between each lioness and the prey determine their duties and hunting groups. The second step of LA mimics this by evaluating each design candidate and selecting the top three candidates which are relatively close to the optimal design. When lionesses encircle the prey, both parties move and chase simultaneously and they alternately change their relative positions towards to the prey. The third step of LA mimics this hunting process by alternating the columns of the top candidate design matrices in order to create new

Figure 3.1: Lionesses Hunting Positions



design candidates for the upcoming iterations. In this hunting process, there are chances that prey would successfully escape. The last step of LA mimics it by probabilisticly enforcing a mechanism for avoiding local optima to make changes on design candidates. The centre solution is returned as the optimal design, which matches the fact that centre lioness is the closest from the prey.

Below, we detail these four steps in LA. We follow the same notations in Section 3.2, where $\mathbf{X}$ denotes a design matrix, k denotes the number of factors and n denotes the number of observations.

The first step of LA is to generate random design candidates, where the number of candidates can be determined by users. The LA starts with multiple candidates, which should lead to a more robust performance. Starting from a single candidate could be easily trapped at some local optimum and it may take undesirably long time to improve the situation starting from a bad candidate.

The second step of LA is to evaluate all design candidates according to an optimality criterion. The candidate with the best performance (global best) will be denoted as the centre solution (CS), the candidate with the second best performance (second global best) will be denoted as the left wing (LW), and the candidate with the third best performance (third global best) will be denoted as the right wing (RW). Unlike other SI algorithms, LA includes the top three candidates, while most other algorithms only consider the best one. The LA adopts the top three candidates since they would share information which can facilitate algorithm not getting stuck at the local optimum. In addition, the top three candidates may be very different from each other, which may expedite the exploration of the solution space.

The third step of LA is to create a new collection of design candidates via CS, LW and RW. Let $X_1^{new}, \ldots, X_m^{new}$ denote the new design candidates, where $m \equiv 6k+3$. Let the first three new candidates be CS, LW and RW, i.e. $X_1^{new} =$ CS, $X_2^{new} =$ LW, and $X_3^{new} =$ RW. Set a count index $C = 4$, and let $X_C^{new} =$ CS. Replace the first column of X_C^{new} with the first column of LW and set $C = C + 1$. After that, let $X_C^{new} =$ CS (currently $C = 5$), replace the first column of X_C^{new} with the first column of RW and let $C = C + 1$. Then (currently $C = 6$), let $X_C^{new} =$ CS, replace the second column of X_C^{new} with the second column of LW and set $C = C + 1$. Similarly, let $X_C^{new} =$ CS (currently $C = 7$),

replace the second column of X_C^{new} with the second column of RW and set $C = C + 1$. Repeat this procedure for each column and we have design candidates $X_4^{new}, \ldots, X_{2k+3}^{new}$. Next, let X_C^{new} = LW (currently $C = 2k + 4$) instead of CS, replace the first column of X_C^{new} with the first column of CS and set $C = C + 1$. Similarly for $C = 2k + 5$, let X_C^{new} = LW, replace the first column of X_C^{new} with the first column of RW and set $C = C + 1$. Repeat such procedures which will lead to design candidates $X_{2k+4}^{new}, \ldots, X_{4k+3}^{new}$. Finally, let X_C^{new} = RW (currently $C = 4k + 4$), replace one column at a time with the corresponding column from CS and LW, and this forms design candidates $X_{4k+4}^{new}, \ldots, X_{6k+3}^{new}$. Now we have the new collection of design candidates. This is the core step for LA, which has several advantages. Different from other popular algorithms, the LA creates new candidates through a column exchanges technique, which enables top candidates share information with each other when creating new ones. In addition, column exchanges can work for both discrete designs and continuous designs, which makes the LA suitable for different design types. The number of new design candidates is $6k + 3$. For designs with small k, the LA does not need too much computational resources. For designs with moderate k and large n ($n >> k$), the LA expands search region in design space to improve the quality of result.

The last step of LA is for avoiding local optima. Let X_1^{new} remain unchanged to preserve the current global best and probabilistically update the elements in each column of candidates $X_2^{new}, \ldots, X_m^{new}$. Here, the mechanisms of updates for discrete designs and continuous designs are different. We will illustrate their details in the next subsection. After the updates, the design candidates will be evaluated again for the next iteration. When the algorithm stops, the centre solution will be returned as the optimal design. Besides avoiding local optima, this mechanism also helps with the exploitation within the algorithm, because it provides opportunities to update element(s) for each column of each new design candidate. It works well together with the column-exchange technique in the core step.

The key idea of LA is to iteratively update and identify CS, LW and RW, and use them to further create and improve other design candidates until the stopping condition is met. We summarize the general framework of LA in Algorithm 4.

Algorithm 4 General Framework of Lioness Algorithm

```
Generate random design candidates
Evaluate each design candidate.
while not converge do
    Determine CS, LW and RW.
    Create new design candidates via CS, LW and RW.
    Apply mechanism for avoiding local optima.
    Evaluate new design candidates.
end while
Determine and return the CS.
```

3.3.2 Implementation for Discrete Designs

In this subsection, we illustrate how to apply the general framework of LA to identify optimal discrete designs including Latin hypercube designs (LHDs) and order-of-addition designs (OofAs) under various optimality criteria defined in Section 3.2.

Latin Hypercube Designs:

Consider the optimal LHDs based on the optimality criterion Φ which can be the space-filling criterion in Equation (3.1), the Maxpro criterion in Equation (3.2) or the orthogonality criterion in Equation (3.3). To identify optimal LHDs, the first step of LA is to generate a user-defined number of random $n \times k$ LHDs, which are denoted as as $X_1, X_2, \ldots$, and we recommend to use the currently available `rLHD(n,k)` function in the `LHD` package (H. Wang et al., 2021). Then, all random LHDs will be evaluated based on the optimality criterion Φ, and the CS, LW and RW will be determined according to the values of $\Phi(X_1), \Phi(X_2), \ldots$. The third step is to use CS, LW and RW to create a new collection of design candidates as illustrated in the general framework. In the last step, we would apply the following mechanism to every column in each of new designs except X_1^{new} (i.e. $X_2^{new}, \ldots, X_m^{new}$). Let $X_{(.j)}^{new}$ be the j^{th} column of a new design X^{new}, and draw a random number z from the standard uniform distribution (aka. Uniform$(0, 1)$). If $z < p*$, where $p*$ is a user-defined tuning parameter, two randomly selected elements within $X_{(.j)}^{new}$ will be switched. Parameter $p*$ is a probability that controls how likely this switch would occur. Note that for LHDs with small k (i.e. $k \leq 3$), it is possible that exchanging columns does not lead to any improvement, while exchanging elements may be more effective. Thus, we recommend setting $p*$ to be $\frac{1}{k-1}$, which guarantees the exchanges will happen when $k = 2$. It becomes less likely to happen when k increases, which further saves CPU time for LHDs with many factors. After all above steps, calculate $\Phi(X_i^{new})$ for $i = 1, \ldots, m$ for the next iteration. When algorithm stops, return the CS as the output (i.e. the identified optimal design).

Order-of-addition Designs:

To identify optimal Order-of-addition designs (OofAs), the LA starts with generating an user-defined number of random n by k OofAs, and we recommend to use the currently available `rOofAC(n,k)` function in the `LA` package (H. Wang et al., 2022). The second step is to find the PWO matrices and moment matrices for each of the random OofAs, and evaluate them based one optimality criterion that were demonstrated in Section 3.2 (i.e. the D-optimality criterion). Note that the algorithm function for OofAs in the `LA` package will calculate all the PWO matrices, moment matrices, and D-optimality criterion automatically, so practitioners do not need to make additional calculations. After the CS, LW and RW are determined, the third step is exactly the same as the steps in the general framework of LA, except that all the column-wise replacements become row-wise instead. For example, when $C = 4$, replace the first row of X_C^{new} with the first row of LW and then set $C = C + 1$ (not column any more). The mechanism for avoiding local optima in the last step is similar to that for identifying optimal LHDs, except that it works in a row-wise manner. For all the new design candidates except for X_1^{new}, every row in each of new designs

will probabilisticly encounter the following mechanism. Let $X_{(i.)}^{new}$ be the i^{th} row of a new design X^{new}, and z and $p*$ are defined the same as above. If $z < p*$, two randomly selected elements within $X_{(i.)}^{new}$ will be switched. For OofAs, the recommended setting for $p*$ would be $\frac{1}{n-1}$ for the same reasons discussed in the LHD case. After all above steps, evaluate all the new design candidates for the next iteration. When the algorithm stops, return CS as the optimal design.

As discussed in Section 3.2, each column of an LHD is a random permutation of $1, 2, \ldots, n$, while each row of an OofA is a random permutation of $1, 2, \ldots, k$. Thus, the optimization for LHDs focuses on columns, while the optimization for OofAs focuses on rows. We provide the pseudo-codes of LA for OofAs in the supplementary materials.

3.3.3 Implementation for Continuous Designs

In this subsection, we illustrate how to apply the general framework of LA to identify optimal continuous designs. In the first step, the algorithm starts with generating a user-defined number of random design candidates. Assume each design candidate has k factors and n observations for each factor and let $[LB_j, UB_j]$ denote the domain of the j^{th} factor with $j = 1, \ldots, k$. The n elements of the j^{th} ($j = 1, \ldots, k$) column in any candidate design are randomly drawn from an uniform distribution within $[LB_j, UB_j]$. In the second step, evaluate each of them based on the optimality criterion Φ and determine the CS, LW and RW. The third step is to use CS, LW and RW to create a new collection of design candidates. Here the second and the third step are the same as the general framework.

The last step is to apply a mechanism for avoiding local optima. For all the new design candidates except for X_1^{new}, we will do the following procedures for every element in each design candidate. Let $X_{(ij)}^{new}$ be an element from the i^{th} row and the j^{th} column of X^{new}. We draw a random number z from the standard uniform distribution, which is defined the same as above. If $z < p$ where p is given by

$$\frac{(\text{total number of iterations} - \text{current iteration} + 1)}{\text{total number of iterations}},$$

$X_{(ij)}^{new}$ will be updated with the following equation

$$X_{(ij)}^{new} = X_{(ij)}^{new} \pm \alpha X_{(ij)}^{new}, \tag{3.4}$$

where $\pm$ is a random draw from $+$ and $-$ with 50% chance for each and α is a tuning parameter to control the variation. Parameter p starts with being 1 and decreases over iterations. This setting of p guarantees that each element can search around its current value at the early stage of the algorithm to identify potentially better designs, which will prevent optimal designs being switched away in the latter stage. When elements exceed the upper/lower bounds of their domain after Equation (3.4), they should be corrected and set to the bound values. Then, calculate $\Phi(X_i^{new})$ for $i = 1, \ldots, m$ for the next iteration. If algorithm stops, determine CS and return it as the optimal design. For complex designs requiring both minimization and maximization, i.e. minimax G-optimal designs, one can implement LA twice (one for minimization and one for maximization) to identify optimal designs. We will further illustrate the details for identifying

G-optimal designs in Section 3.4. In addition, we will provide a study on the tuning parameter α in Section 3.4 as well.

Here we provide an illustrative example. Consider a health scientist is designing an experiment on 20 rats to discover the relationship between cooking gas and lung function. The response variable is the maximum lung volume, and the explanatory variable is the time per day (in hours) exposed to a given level of cooking gas in the atmosphere. Assume a simple linear regression model is implemented and the goal is to find the 20-run A-optimal design with domain $[0, 24]$. In simple linear regression model, the A-optimal design is obtained by minimizing the trace of the inverse of information matrix, which can be expressed by

$$\Phi = \frac{20 + \sum_{i=1}^{20} x_i^2}{20 \sum_{i=1}^{20} x_i^2 - (\sum_{i=1}^{20} x_i)^2},$$

where the 20 x's stand for the 20 runs in the design matrix. Hence, the optimal setting of x's that minimizes above criterion is the A-optimal design. Next we show how LA can identify the optimal design for this example step by step. First, the LA starts with generating random design candidates, where each candidate is a 1 by 20 vector. Based on the domain for the factor, we have $LB_1 = LB_2 = \ldots = LB_{20} = 0$ and $UB_1 = UB_2 = \ldots = UB_{20} = 24$. Each element in each candidate design would be a random draw from Uniform$(0, 24)$. For example, one candidate may look like $(15.8, 2.4, 22.6, \ldots, 4.1)$. The second step is to evaluate each candidate via the A-optimality criteria above and determine the CS, LW and RW. Suppose they are equal to the following: CS $= (19.5, 17.3, 0, \ldots, 19.7)$, LW $= (8.1, 1.7, 3.4, \ldots, 0.8)$, and RW $= (7.1, 14.9, 7.4, \ldots, 10.3)$. The third step is to create a new collection of design candidates with the following technique. Let $X_1^{new} =$ CS, $X_2^{new} =$ LW, and $X_3^{new} =$ RW. Set a count index $C = 4$, let $X_{C=4}^{new} =$ CS and replace the first column of X_4^{new} with the first column of LW, i.e. $X_{C=4}^{new} = (8.1, 17.3, 0, \ldots, 19.7)$, then $C = C + 1$. Let $X_5^{new} =$ CS and replace the first column of X_5^{new} with the first column of RW, i.e. $X_{C=5}^{new} = (7.1, 17.3, 0, \ldots, 19.7)$, then $C = C+1$. X_6^{new} and X_7^{new} follows the same pattern but replace the second column instead, i.e. $X_{C=6}^{new} = (19.5, 1.7, 0, \ldots, 19.7)$ and $X_{C=7}^{new} = (19.5, 14.9, 0, \ldots, 19.7)$. Continue this procedure all the way to last column, i.e. $X_{C=42}^{new} = (19.5, 17.3, 0, \ldots, 0.8)$ and $X_{C=43}^{new} = (19.5, 17.3, 0, \ldots, 10.3)$. For new design candidates X_{44}^{new}, ..., X_{83}^{new}, they should be created based on LW with the same logic, i.e. $X_{C=44}^{new} = (19.5, 1.7, 3.4, \ldots, 0.8)$, $X_{C=45}^{new} = (7.1, 1.7, 3.4, \ldots, 0.8)$, ..., $X_{C=82}^{new} = (8.1, 1.7, 3.4, \ldots, 19.7)$, and $X_{C=83}^{new} = (8.1, 1.7, 3.4, \ldots, 10.3)$. Similarly for X_{84}^{new}, ..., X_{123}^{new}, they should be created based on RW with the same logic, i.e. $X_{C=84}^{new} = (19.5, 14.9, 7.4, \ldots, 10.3)$, $X_{C=85}^{new} = (8.1, 14.9, 7.4, \ldots, 10.3)$, ..., $X_{C=122}^{new} = (7.1, 14.9, 7.4, \ldots, 19.7)$, and $X_{C=123}^{new} = (7.1, 14.9, 7.4, \ldots, 0.8)$. In the last step, for $X_2^{new}, \ldots, X_{123}^{new}$, apply the mechanism for avoiding local optima. For example, suppose that one element is 0.90 in one candidate X^{new} and a random draw z from Uniform$(0, 1)$ is smaller than the current p, this element may change to either 0.81 or 0.99 if tuning parameter α is set to be 0.1. After the algorithm stops, LA should return the most recent CS as the A-optimal design. A pseudo-code of LA for continuous designs is outlined in Algorithm 5.

Algorithm 5 Lioness Algorithm for Continuous Designs

Generate random design candidates .
Evaluate each design candidate.
while not converge **do**
  Determine CS, LW and RW.
  Let $X_1^{new} =$ CS, $X_2^{new} =$ LW, and $X_3^{new} =$ RW, and set a count index $C = 4$.
  for each column j of X^{new} **do**
    Let $X_C^{new} =$ CS; replace the j^{th} column of X_C^{new} with the j^{th} column of LW; $C = C + 1$
    Let $X_C^{new} =$ CS; replace the j^{th} column of X_C^{new} with the j^{th} column of RW; $C = C + 1$
  end for
  for each column j of X^{new} **do**
    Let $X_C^{new} =$ LW; replace the j^{th} column of X_C^{new} with the j^{th} column of CS; $C = C + 1$
    Let $X_C^{new} =$ LW; replace the j^{th} column of X_C^{new} with the j^{th} column of RW; $C = C + 1$
  end for
  for each column j of X^{new} **do**
    Let $X_C^{new} =$ RW; replace the j^{th} column of X_C^{new} with the j^{th} column of CS; $C = C + 1$
    Let $X_C^{new} =$ RW; replace the j^{th} column of X_C^{new} with the j^{th} column of LW; $C = C + 1$
  end for
  for each X^{new} (except X_1^{new}) **do**
    for each row i of current X^{new} **do**
      for each column j of current X^{new} **do**
        if $z < p$ **then**
          $X_{(ij)}^{new} = X_{(ij)}^{new} \pm \alpha \times X_{(ij)}^{new}$
        end if
      end for
    end for
  end for
  Evaluate all the X^{new}s.
end while
Determine and return the CS.

Table 3.1: Average CPU Time (in seconds) for the Simulation in Figure 3.2.

	30×3	40×4	50×5	60×6
LA	0.360	0.974	2.164	4.138
SLHD	0.006	0.012	0.015	0.017

3.4 Performance of the Proposed Algorithm

In this section, we provide simulation results to assess the performance of the proposed LA for identifying different types of designs. This algorithm for each design type can be directly implemented by our developed R package `LA`. All the CPU time in this section are based on Intel Core i7-10750H 2.60GHz processor.

3.4.1 Numerical Results on Latin Hypercube Designs

In this subsection, simulations were conducted for three types of LHDs: maximin distance LHDs, maximum projection LHDs, and nearly orthogonal LHDs. Popular search algorithms were compared with LA. For each design size, we run every search algorithm multiple times and present results along with CPU time.

Maximin Distance LHDs

We use the ϕ_p criterion defined in Equation (3.1) under the L_2 distance to compare performances. One of the most efficient search algorithms for maximin distance LHDs is SLHD (Ba et al., 2015), which can be implemented from the `SLHD` package (Ba, 2015). We will compare the LA with this current popular algorithm.

Figure 3.2 is the boxplots of ϕ_p values under the L_2 distance for both LA and SLHD algorithms with different design sizes. For each design size, we set the maximum number of iterations to be 500 and the rest algorithm settings remain to the default. We run every algorithm 100 times. All the panels in Figure 3.2 show that the LA algorithm outperforms the SLHD algorithm as the LA has smaller ϕ_p values, and this is consistent when design sizes increase. Table 3.1 displays the average CPU time (in seconds) for each of the design sizes in Figure 3.2. For design sizes 30×3 and 40×4, both the LA algorithm and the SLHD algorithm has CPU time less then one second. For design sizes 50×5 and 60×6, the CPU time of the LA algorithm increased due to the number of factors increased. This is reasonable since the core step of the LA depends on factor sizes. An increase in factor size would lead more design candidates generated during the searching process. Exploring more design space when seeking the optimal designs for large factor sizes would cost more CPU time in general.

Figure 3.2: Boxplots of ϕ_p Values for LA and SLHD with the Rule of Thumb Sizes.

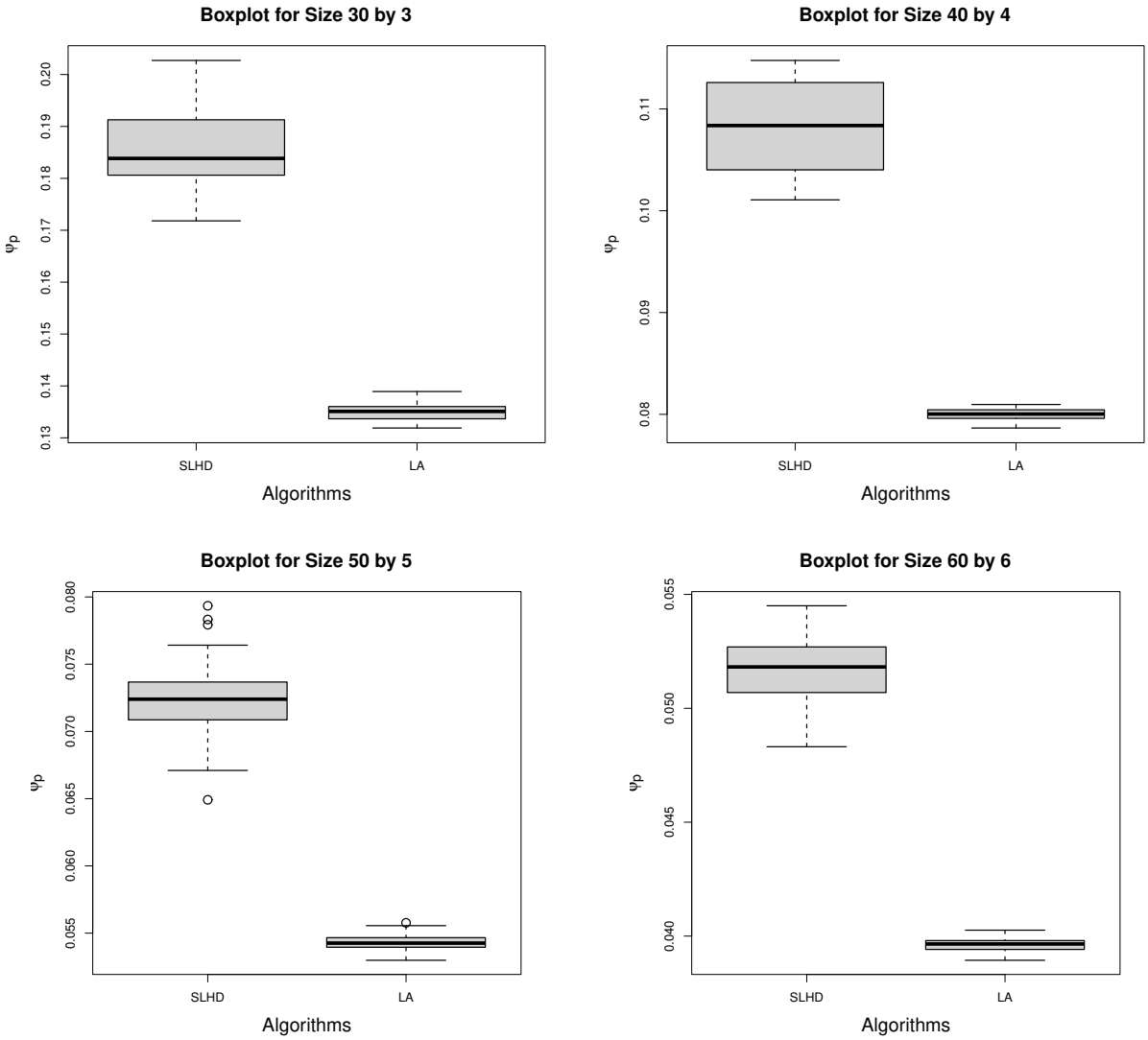


Figure 3.3: Convergence Curves for LA and SLHD with the Rule of Thumb Sizes.

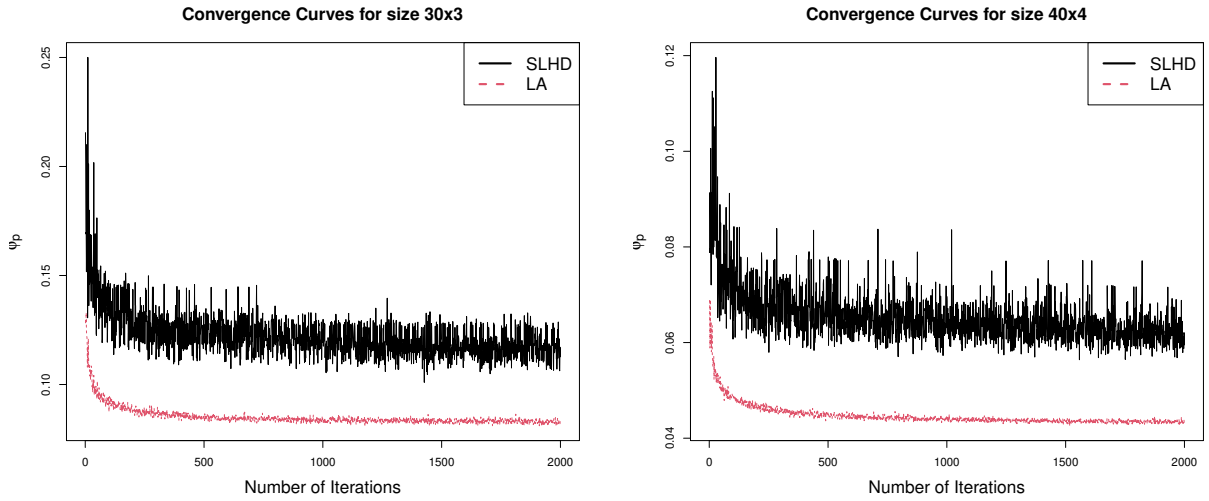
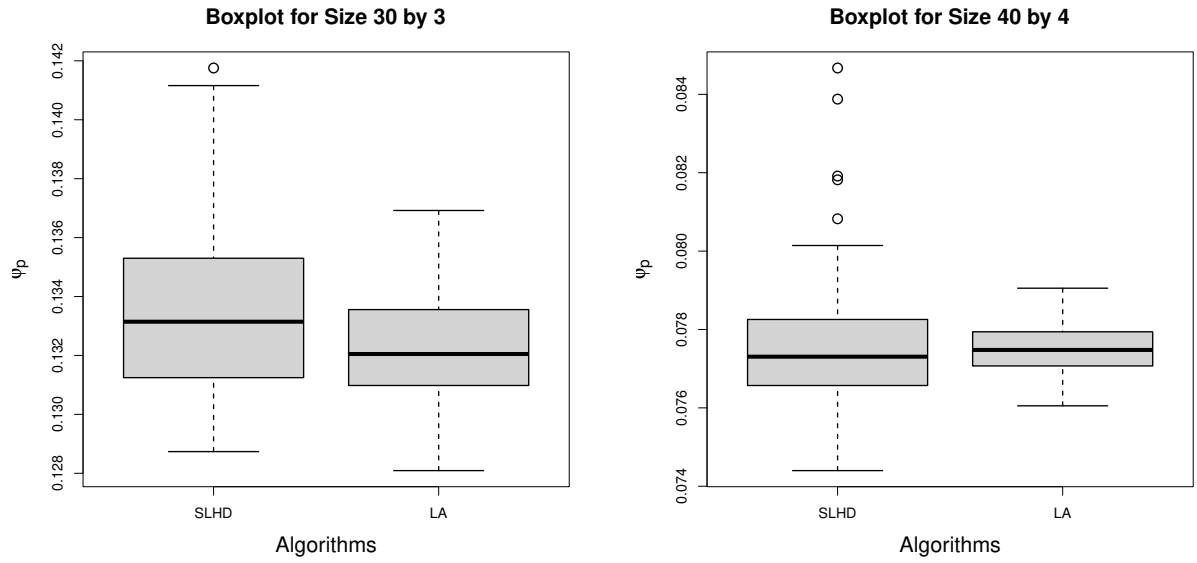


Figure 3.3 shows the convergence curves for both LA and SLHD algorithms with different design sizes. For each design size, we set the maximum number of iterations to be $2,000$ to observe the convergence situation and the stability for each algorithm. For both design sizes 30×3 and 40×4, the LA algorithm has better convergence situation than the SLHD algorithm in terms of smaller ϕ_p values. The SLHD algorithm has less stability than the LA algorithm due to the very unstable convergence curves. This again supports that the LA outperforms the SLHD when they have the same number of iterations.

Next, we no longer fix the maximum number of iterations, that is, we only specify design sizes and let all the algorithm settings remain to the default. Note that the default maximum number of iterations for the SLHD algorithm is 1×10^6, while the default maximum number of iterations for the LA algorithm is $5,000$. Figure 3.4 shows the ϕ_p values under the L_2 distance for both algorithms with design size being 30×3 and 40×4. We run each algorithm 100 times. The left panel is the result for design size 30×3 and the right panel is the result for design size 40×4. The left panel shows the performance of the LA is better than SLHD since the the ϕ_p values are smaller, and the boxplot of the LA is less-stretched. The right panel shows both algorithms found similar median, while the SLHD has smaller ϕ_p values but with more stretched boxplot. With all the default setting, where the SLHD has 1×10^6 iterations and the LA only has $5,000$ iterations, the LA is still very competitive and it outperforms the SLHD when for deisgn size 30×3. Figure 3.4 again shows that the LA has better stability than the SLHD. For the CPU time, it took the SLHD 0.090 second to finish the 1×10^6 iterations and it took the LA 3.545 seconds to finish the $5,000$ iterations when the design size is 30×3. For design size 40×4, it took the SLHD 0.190 second

Figure 3.4: Boxplots of ϕ_p Values for LA and SLHD with the Rule of Thumb Size.



to finish the 1×10^6 iterations and it took the LA 9.462 seconds to finish the $5,000$ iterations. SLHD has the advantage in terms of CPU time.

Together taken, the LA outperforms the SLHD in identifying the maximin distance LHDs when they have the same number of iterations, and the LA has better convergence situation and stability. If there is no limitations on the maximum number of iterations, both algorithm performs similarly and the LA is slightly better. For computational cost, the SLHD has lower CPU time at all cases.

Maximum Projection LHDs

We consider identifying maximum projection LHDs (MaxPro LHDs) under the the maximum projection criterion ψ defined in Equation (3.2). Joseph et al., 2015 proposed the maximum projection criterion along with a search algorithm based on simulated annealing (Morris & Mitchell, 1995) in their work. This algorithm, which is denoted as MaxProLHD, can be implemented from the `MaxPro` package (Ba & Joseph, 2018). We will compare the LA with this popular algorithm.

Here we only specify design sizes and let all the algorithm settings remain to their default. Note that the default maximum number of iterations for the MaxProLHD algorithm is 1×10^6, while the default maximum number of iterations for the LA algorithm is $5,000$. For each design size, we run each algorithm 100 times. Figure 3.5 is the boxplots of ψ Values for LA and MaxProLHD with the Rule of Thumb Sizes. For all the cases in Figure 3.5, LA identified better designs because of the smaller ψ values. Table 3.2 displays

Table 3.2: Average CPU Time (in seconds) for the Simulation in Figure 3.5.

	40×4	50×5	60×6	70×7
LA	0.627	1.285	2.417	4.205
MaxProLHD	1.726	3.066	4.980	7.060

Table 3.3: Minimum max$|q|$ and Average CPU time (in seconds) under Different Design Sizes.

	20×2	20×4	40×4	40×8	60×6	80×8	80×12	120×12	160×16
max$\|q\|$	0	0.002	0.001	0.003	0	0.001	0.003	0.001	0.003
CPU	0.114	0.333	0.551	2.079	1.192	2.867	8.401	10.982	29.851

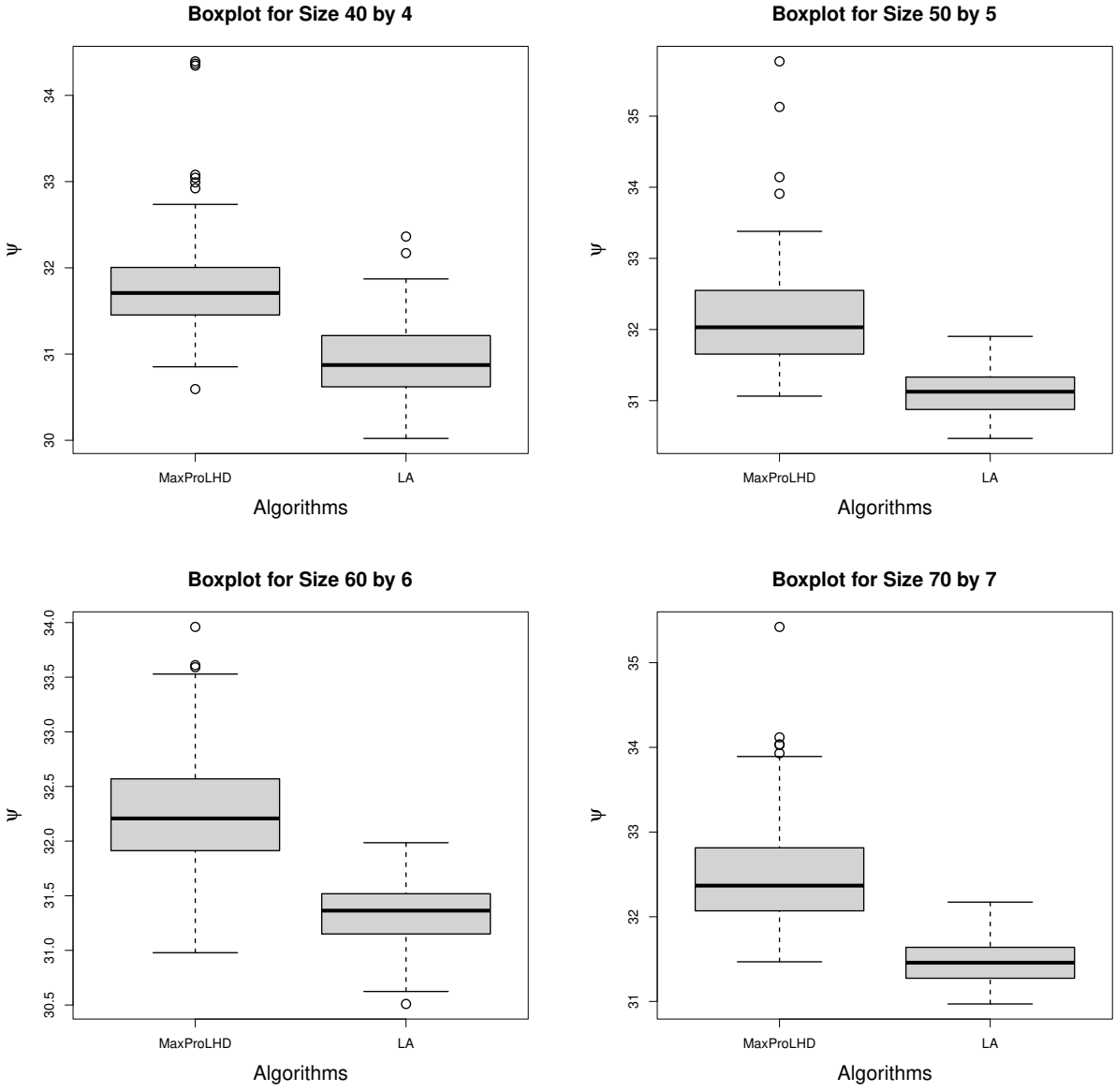
the average CPU time (in seconds) for each of the design sizes in Figure 3.5. The LA has less CPU time than MaxProLHD in all the cases. If the LA has better performance than MaxProLHD when the LA only has $5,000$ iterations and the MaxProLHD has 1×10^6 iterations, then the LA would definitely outperforms the MaxProLHD if they have the same maximum number of iterations. For the same reason, the LA would have better convergence situation as well.

Orthogonal and Nearly Orthogonal LHDs

For orthogonal and nearly orthogonal LHDs (OLHDs and NOLHDs), we adopt the maximum absolute correlation criterion, the max$|q|$ defined in Equation (3.3). OLHDs (or NOLHDs) can be generated by algebraic construction methods (Butler, 2001; Cioppa & Lucas, 2007; C. D. Lin et al., 2009; Sun et al., 2010; Ye, 1998) with almost no computational cost for certain design sizes. Therefore we use the LA to identify OLHDs and NOLHDs with design sizes where theoretical results are not available. For each design size in the simulation, we run the LA for 100 times and recorded the best design found to evaluate its performance on how close it can achieve when comparing with the 0 column-wise correlation.

Table 3.3 shows the minimum max$|q|$ values under different design sizes along with average CPU Time (in seconds) for the LA. For design sizes 20×2, 60×6, the LA found the OLHD as the max$|q|$ values are zeros. For the rest cases in the Table 3.3, the LA is able to identify NOLHDs with maximum absolute correlations smaller or equal to 0.003. For design sizes 80×8 or smaller, the CPU time of the LA is below 3 seconds. For relative large design sizes, the CPU time of the LA is also reasonable (11 seconds for 120×12 and 30 seconds for 160×16). The simulations for OLHDs and NOLHDs again demonstrate the novelty of the LA where the number of design candidates is factor-dependent. For small factor size designs, the number of design candidates is relatively small which saves CPU time. For large factor size designs, the number of design candidates is relatively large to explore more design space to guarantee the quality of result.

Figure 3.5: Boxplots of ψ Values for LA and MaxProLHD with the Rule of Thumb Sizes.



3.4.2 Numerical Results on Order-of-Addition Designs

In this subsection, simulations were conducted for identifying D-optimal order-of addition designs (OofAs) under the pairwise-order (PWO) model. For a given factor size k, the full OofA (run size n equals to $k!$) is the optimal design (D. K. Lin & Peng, 2019). However, often times there is no enough budget to conduct the full experiment. For example, $k = 10$ yields more than 3.6 million distinct runs, and it is almost impossible to experiment every of them, not to mention for even large factor size k. Hence, an efficient subset of a full OofA that is almost as good as the full experiment but with much less runs would be a more preferred choice.

Finding the most efficient subset of a full OofA can be difficult. For example, consider a 5-factors OofA, where the full OofA has $5! = 120$ distinct runs. If the budget only supports 20 runs, there would be approximately $\binom{120}{20} = 2.946 \times 10^{22}$ possible experiments to choose from. To evaluate how good an OofA is, we use D-efficiency, which is calculated as the relative between a subset and its full OofA, and a larger D-efficiency indicates a better design. Note that an optimal design has a D-efficiency of 1. We will implement the proposed LA to identify D-optimal OofAs on different design sizes, and evaluate the performance of the LA based on the D-efficiencies. For each design size, we run the LA for 100 times, record the best design found along with CPU time, and calculate the D-efficiency. Results are summarized in Table 3.4.

Table 3.4 shows the D-efficiencies and Average CPU time (in seconds) of LA with different sizes for OofAs. When factor size is 4, the number of distinct runs is 24. For design size 12×4, the LA identified a D-optimal design with 100% D-efficiency, which is as good as the full OofA but with only half of its runs, and the CPU time is less than 3 seconds. For the cases of 10×4 and 11×4, D-efficiencies decreased to 63.1% and 76.6%, respectively. If the computational budget cannot support 12 runs for 4 factors, we can see that the trade-off between D-efficiencies and design sizes can be very sensitive. When factor size is 5, the number of distinct runs becomes 120. For all the design sizes listed in Table 3.4 with factor size 5, designs identified by the LA has D-efficiencies greater than 92%. For design size 60×5, the LA identified a D-optimal design with 100% D-efficiency, which is as good as the full OofA but with only half of its runs, and the CPU time is less than 2 minutes. If the computational budget is very limited, the design size 40×5 could be a good option since the CPU time is less than half of the CPU time of size 60×5 and the D-efficiency is 92.6%. When factor size is 6, the number of distinct runs increases to 720. For design sizes 60×6, 65×6, and 70×6, designs identified by the LA has D-efficiencies greater than 90%. Especially for the case of 60×6, it is worthy to notice that 60 runs is approximately 8.33% of the full OofA who has 720 runs, but the D-efficiency is as good as 94.6%. The corresponding CPU time is less than 3 minutes. Together taken, this subsection proves the capability of the LA in identifying D-optimal OofAs with much less run sizes while having high D-efficiencies at the same time.

3.4.3 Numerical Results on Continuous Designs

In this subsection, simulations were conducted for identifying optimal continuous designs. We provide one application to show the superiority of the LA compared to some other popular methods in existing

Table 3.4: D-efficiencies and Average CPU time (in seconds) of LA with different sizes for OofAs

Factor Sizes	Maximum Runs	Design Sizes	D-efficiencies	CPU
$k = 4$	$4! = 24$	10×4	63.1%	1.977
		11×4	76.6%	2.380
		12×4	100%	2.764
		13×4	90.4%	3.113
		14×4	85.2%	3.577
$k = 5$	$5! = 120$	40×5	92.6%	45.886
		45×5	94.1%	56.890
		50×5	96.4%	70.577
		55×5	95.9%	85.407
		60×5	100%	101.277
$k = 6$	$6! = 720$	50×6	88.3%	115.467
		55×6	86.8%	138.297
		60×6	94.6%	165.215
		65×6	90.9%	190.708
		70×6	93.9%	226.028

literature on G-optimal designs, and we provide another application to show the flexibility of the LA along with a study on the tuning parameter α.

G-optimal Design under Second-order Polynomial Model

Consider a second-order polynomial model:

$$y = \beta_0 + \sum_{i=1}^{k} \beta_i x_i + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \beta_{ij} x_{ij} + \sum_{i=1}^{k} \beta_{ii} x_{ii}^2 + \epsilon,$$

where y is the response variable, x's are the explanatory variables, β's are the unknown parameters, and ϵ is the error term that is independently and normally distributed with a constant variance. The domain for each x is assume to be $[-1, 1]$. Suppose there are n observations ($n \geq \binom{k+2}{2}$) and the model can be written as

$$\mathbf{y} = \mathbf{X}_m \boldsymbol{\beta} + \boldsymbol{\epsilon},$$

where $\mathbf{y} = [y_1, y_2, \ldots, y_n]^T$, $\boldsymbol{\beta} = [\beta_0, \beta_1, \ldots, \beta_k, \beta_{12}, \ldots, \beta_{k-1,k}, \beta_{11}, \ldots, \beta_{kk}]^T$, $\boldsymbol{\epsilon} = [\epsilon_1, \epsilon_2, \ldots, \epsilon_n]^T$, and the model matrix, $\mathbf{X}_m$, is

$$\mathbf{X}_m = \begin{bmatrix} 1 & x_{11} & x_{12} & \ldots & x_{1k} & x_{11}x_{12} & x_{11}x_{13} & \ldots & x_{1,k-1}x_{1k} & x_{11}^2 & x_{12}^2 & \ldots & x_{1k}^2 \\ 1 & x_{21} & x_{22} & \ldots & x_{2k} & x_{21}x_{22} & x_{21}x_{23} & \ldots & x_{2,k-1}x_{2k} & x_{21}^2 & x_{22}^2 & \ldots & x_{2k}^2 \\ \ldots & & & & & & & & & & & & \\ 1 & x_{n1} & x_{n2} & \ldots & x_{nk} & x_{n1}x_{n2} & x_{n1}x_{n3} & \ldots & x_{n,k-1}x_{nk} & x_{n1}^2 & x_{n2}^2 & \ldots & x_{nk}^2 \end{bmatrix}.$$

The design matrix for the model, $\mathbf{X}$, is the second to the $(k+1)th$ column of the model matrix:

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1k} \\ x_{21} & x_{22} & \ldots & x_{2k} \\ \ldots & & & \\ x_{n1} & x_{n2} & \ldots & x_{nk} \end{bmatrix},$$

and the elements of design matrix determine the elements of model matrix. Let $\mathbf{x}^T$ denote a prediction point of the model, where $\mathbf{x}^T = [1, x_1, x_2, \ldots, x_k, x_1x_2, x_1x_3, \ldots, x_{k-1}x_k, x_1^2, x_2^2, \ldots, x_k^2]$. The scaled prediction variance (Borkowski, 2003; Hernandez & Nachtsheim, 2018) at $\mathbf{x}^T$ is given by the following:

$$n\mathbf{x}^T(\mathbf{X}_m^T\mathbf{X}_m)^{-1}\mathbf{x},$$

where $\mathbf{X}_m^T\mathbf{X}_m$ is the information matrix for the model based on $\mathbf{X}$. For an arbitrary design matrix, there exist an prediction point, $\mathbf{x}^*$, that maximizes the scaled prediction variance (that is, the worst possible prediction) such that

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\, n\mathbf{x}^T(\mathbf{X}_m^T\mathbf{X}_m)^{-1}\mathbf{x}.$$

The G-optimal design aims to minimize the maximum scaled prediction variance (that is, the design has the relative "best" worst possible prediction), which constitutes a minimax problem. Let $\mathbf{X}_G^*$ denote the G-optimal design, which can be defined as

$$\mathbf{X}_G^* = \underset{\mathbf{X}}{\operatorname{argmin}}\, \underset{\mathbf{x}}{\operatorname{argmax}}\, n\mathbf{x}^T(\mathbf{X}_m^T\mathbf{X}_m)^{-1}\mathbf{x}. \tag{3.5}$$

To identify G-optimal designs and solve the minimax problem, we can implement LA twice (one for outer loop and one for inner loop). Outer loop aims to optimize design candidate matrices, and inner loop aims to provide the maximum scaled prediction variance for each design candidate over iterations. For example, starting with outer loop, LA first generates a certain number of design candidate matrices. For each candidate, implement a separate LA for searching and identifying the optimal setting of $\mathbf{x}^*$ that maximizes the scaled prediction variance (this is the inner loop). After having each candidate's possible maximum scaled prediction variance, CS, LW and RW will be determined accordingly. The rest procedure should be exactly the same as the demonstration in Section 3.3.3. Here we provide some simulation results of G-optimal designs identified by LA for two factors cases ($k = 2$) along with CPU time (in seconds) in

Table 3.5: Relative G-efficiency for Different Algorithms with Respect to GA (Borkowski, 2003).

Design Sizes	$G(I_\lambda)$-CEXCH	G-CEXCH	GA	LA	CPU of LA
6×2	96.1%	98.6%	100%	120.9%	64.123
7×2	95.5%	97.9%	100%	116.6%	65.394
8×2	94.7%	99.7%	100%	110.5%	69.397
9×2	95.8%	97.0%	100%	125.9%	75.629
10×2	93.2%	97.5%	100%	117.1%	78.078
11×2	97.0%	94.0%	100%	100.5%	80.571
12×2	95.1%	101.2%	100%	106.7%	84.490

Table 3.5. For each design size, we run the LA for 100 times and recorded the best design found. Results in existing literature (Borkowski, 2003; Hernandez & Nachtsheim, 2018) will be used to compare with the designs identified by LA.

In Table 3.5, the $G(I_\lambda)$-CEXCH and G-CEXCH are two popular algorithms by Hernandez and Nachtsheim, 2018, and the GA is another widely used algorithm by Borkowski, 2003. Here, all the relative efficiencies are calculated based on the results from GA. If the ratio is smaller than one, the performance of the current algorithm is less G-efficient than the GA. The results of $G(I_\lambda)$-CEXCH, G-CEXCH and GA can be found in Hernandez and Nachtsheim, 2018. For all the design sizes listed in Table 3.5, the LA identified better G-optimal designs than those from all other algorithms. The G-efficiencies are over 120% for design sizes 6×2 and 9×2, G-efficiencies are more than 110% for design sizes 7×2, 8×2 and 10×2, and G-efficiencies are greater than 100% for 11×2 and 12×2. The average CPU time for different design sizes range from 64.123 seconds to 84.490 seconds, which are all between 1 and 1.5 minutes. We list the G-optimal designs found by LA for all design sizes here in supplementary materials for reference.

D-optimal Design under Linear Regression Model and a Discussion on the Tuning Parameter α

Recall the illustrative example in Section 3.3.3. A health scientist is designing an experiment on 20 rats to discover the relationship between cooking gas and lung function. The response variable is the maximum lung volume, and the explanatory variable is the time per day (in hours) exposed to a given level of cooking gas in the atmosphere. The assumption is that a simple linear regression model is implemented. For example,

$$y = \beta_0 + \beta_1 x + \epsilon,$$

where y is the response variable, x is the explanatory variables, β's are the unknown parameters, and ϵ is the error term that is independently and normally distributed with a constant variance. The domain for x is $[0, 24]$ to indicate time per day (in hours) exposed. Since there are 20 rats in the experiment, the model can be written as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon},$$

where $\mathbf{y} = [y_1, y_2, \ldots, y_{20}]^T$, $\boldsymbol{\beta} = [\beta_0, \beta_1]^T$, $\boldsymbol{\epsilon} = [\epsilon_1, \epsilon_2, \ldots, \epsilon_{20}]^T$, and the design matrix, $\mathbf{X}$, is

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \ldots & \ldots \\ 1 & x_{20} \end{bmatrix}.$$

This time we assume the goal is to find the 20-run D-optimal design. After knowing the design size and domains for each column, we can directly implement the LA to identify the optimal design. For example, the following code would immediately return the 20-run D-optimal design to above problem: `X=LA_OptC(n=20,lb=c(1,0),ub=c(1,24),OC="D");round(X,4)`. Note that the way how LA works on multiple linear regression is very similar to that on simple linear regression (simply add more column domains to the lower and upper bound option).

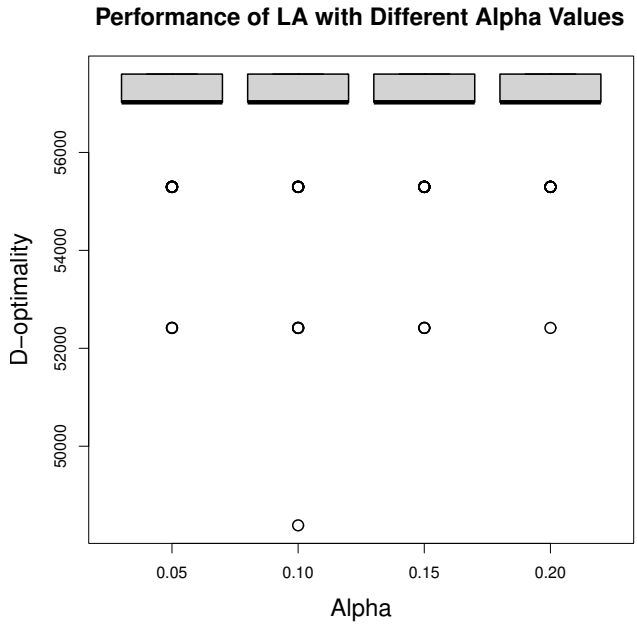
Next, we use above example to study the effect of tuning parameter α. We implement the LA with 4 different α values, which are 0.05, 0.1, 0.15, and 0.2, respectively. For each α value, we run the LA for 100 times. Figure 3.6 is the boxplots of D-optimality criterion (inverse of the determinant of information matrices) for designs identified by LA with different α values. Figure 3.6 shows that all the 4 cases have similar performance, except when $\alpha = 0.10$, the algorithm identified a better design than the rest cases. This is only one of many examples can be used to study the effect of tuning parameter α, and the conclusion here may not necessarily workable for every situation. Reasonable choice of α can be highly depend on problem itself. Generally speaking, relatively large α values are good for exploration problems, while relatively small α values are good for exploitation problems. Based on our empirical observations, $\alpha = 0.1$ works for most of the cases and balances exploration and exploitation.

3.5 Discussion

In this chapter, we propose a novel meta-heuristics optimization algorithm, called the lioness algorithm (LA), for finding optimal design of experiments including the maximin distance LHDs, the maximum projection LHDs, the orthogonal LHDs, the optimal order-of-addition designs, and the D-optimal as well as G-optimal designs. The LA embeds a column exchange technique, which makes it efficient for finding both discrete designs and continuous designs. The LA strengthens the exploration via including multiple global best solutions during the searching process. The mechanism for avoiding local optimum can be customized to address practical considerations, which provides flexibility to practitioners. It requires less memory and computing time, and usually leads to better results compared to some currently popular methods in the literature. We develop efficient implementations of the LA for both continuous and discrete optimal designs and integrate them into the R package `LA`.

Inspired by the three groups of hunting lionesses in nature, the LA involves three global best solutions, which works well in general. It is possible to include more than three solutions to further improve the performance. Yet, it will lead to a larger number of design candidates at each iteration and hence will

Figure 3.6: Boxplots of D-optimality Criterion for LA with Different α Values.



increase the computing time. For other tuning parameters in the LA including the number of initial random design candidates and the parameter α (for identifying continuous designs), users need to set their values depending on the design size and available computing resources. A large number of initial random design candidates can slow down the algorithm, while a small number of them may negatively affect the performance. In this work, we recommend 100 initial random designs and $\alpha = 0.1$. A more thorough study on parameter tuning of the LA can be an important future work.

The search process of LA is stochastic and the LA stops after reaching a maximum number of iterations or CPU time. Like some other meta-heuristics algorithms, we do not have a rigorous proof of its convergence. Another important future work is to study the convergence of the proposed LA under certain assumptions. Though the LA can be used to generate various types of optimal designs, it is not suitable for finding optimal mixture experiments, where each of the factors is an independent proportion of ingredients to form a mixture (Cornell, 1973). For examples of mixture experiments, consider the lipophilic drug solubility and dissolution study on the effect of microemulsion formulation composite (Furlanetto et al., 2011) and simplex screening design on the effects of ten motor octane components (Snee & Marquardt, 1976). Since every row of the design matrix for a mixture experiment has to sum up to 100%, this becomes a constrained optimization problem, and the column exchange technique of LA is

not equipped to incorporate this characteristic of a mixture experiment. Incorporating constraints in LA would be another topic of future research.

3.6 Research Outcomes

In this research, I invented a novel optimization algorithm for identifying optimal design of experiments, which is compatible with different design types. Previous experience with understanding algorithm structure and theories, effectively coding algorithm, and publishing R library all cumulatively contributed to this research. In addition, I learned how to write C++ code via R platform, which is a life-time skill.

Chapter 4

Identification of Predictive MRI and Functional Biomarkers in a Pediatric Piglet Traumatic Brain Injury Model

4.1 Introduction

In United States, traumatic brain injury (TBI) is a leading cause of long-term and permanent disability and even death. Each year, approximately 2.8 million people suffer from a TBI in the United States (Taylor et al., 2017). Among those injuries, $50,000$ people die, $282,000$ people are hospitalized, and 2.5 million people are admitted to the emergency room (ER) making TBI a major healthcare concern. Unfortunately, one of the largest groups affected by TBI are children between the age of 0 and 14 with almost half a million ($473,947$) children going to the ER each year (Faul et al., 2010). Many of these injuries are due to car accidents, sports and other daily activities. After a patient suffers a TBI, one of the major questions has become how to reliably predict functional outcomes and recovery with significant efforts being made in the use of non-invasive imaging modalities including magnetic resonance imaging (MRI) (Galloway et al., 2008; Guild & Levine, 2015).

MRI is widely used in TBI patients and has significant potential to identify key changes in pathophysiology that can be utilized to predict functional outcomes in patients (Cooper et al., 2014; Saatman et al., 2008). MRI is capable of assessing TBI progression longitudinally and providing comprehensive information on spatiotemporal parameters such as lesion volume and midline shift (Chastain et al., 2009; Lee et al., 2012). In humans, TBI tissue damage detected by MRI was found to be related with gait and balance deficits (Caeyenberghs et al., 2010; Drijkoningen et al., 2017). In pediatric TBI patients, regional gray matter volume was positively correlated with gait control as indicated by loss of gray matter volume coinciding with increased step length asymmetry (Drijkoningen et al., 2017). Decreased white matter integrity has also been found to be highly correlated with loss of postural control and balance (Caeyen-

berghs et al., 2009). This suggests that these MRI "biomarkers" might be a powerful tool to predict injury recovery.

Rodent models are widely used to study TBI pathophysiology, motor function deficits and to perform safety and efficacy studies for potential treatments (Marklund, 2016). However, failure to develop a Food and Drug Administration approved treatment despite numerous clinical trials, with more promising therapeutics failing to reach primary end points in phase III trials, has led to increased interest in more translational and predictive animal models (Jain, 2019). The pig has become a TBI model of significant interest due to similarities in brain anatomy, physiology and development (Conrad et al., 2012; Costine et al., 2015; Dobbing & Sands, 1979; Flynn, 1984; Gieling et al., 2011; Paredes et al., 2016). To date, there are only a limited number of studies that assess TBI pathophysiology utilizing MRI in pig models. T1- and T2-weighted sequences are the most commonly used MRI sequences to measure lesion volumes in the TBI pig model (Duhaime et al., 2003; Grate et al., 2003; Karlsson et al., 2019; Rosenthal et al., 2008) and diffusion tensor imaging (DTI) and diffusion-weighted imaging (DWI) have been used to measure white matter integrity and brain microstructure (Conrad & Johnson, 2015). However, there are no pig studies that have taken a quantitative statistical approach to assessing the longitudinal relationship between MRI observed TBI pathology (e.g. lesion volume) and motor function changes in a pediatric pig model.

The objective of this study was to utilize advanced statistical approaches to identify key MRI parameters that are longitudinally associated with changes in motor function in a piglet TBI model. In this study, we utilized principal component analysis (PCA) on 6 MRI parameters and 12 gait parameters collected in a controlled cortical impact (CCI) TBI piglet model to identify parameters that explain the most variability of the data. Furthermore, this information was used to determine if MRI biomarkers can be used to predict motor function impairments via linear regression analysis. This study provides an important first step in understanding the predictive power of MRI biomarkers and motor function outcomes in a pediatric TBI model that more closely resembles human TBI patients than traditional rodent models.

4.2 Materials and Methods

4.2.1 Animals and Controlled Cortical Impact Surgery

Data that underwent advanced statistical analysis was collected as part of a previously published study (Kinder et al., 2019). Briefly, all work was performed in accordance with the University of Georgia Institutional Animal Care and Use Committee guidelines. Eighteen commercially bred, castrated male pigs were born at the Large Animal Research Unit at the University of Georgia. Piglets between 4-5 weeks of age underwent TBI induction surgery. On the day of surgery, piglets were anesthetized using 5% isoflurane and oxygen via surgical mask and then maintained under anesthesia during surgical procedures using 2.5-3% isoflurane. Vitals (temperature, heart rate, and respiration rate) were monitored every 5-10 minutes during anesthesia. Piglets were administered Flunixin (2.2 mg/kg) and butorphanol (0.2 mg/kg) as a pre-med for analgesia. The surgery site was prepared routinely for surgery. The surgical site was clipped, sterilized using Betadine and 70% ethanol, and covered with a sterile drape. A 4cm left-sided incision was

made over the top of the cranium to expose the underlying skull. A craniectomy, approximately 20mm in diameter, was performed using an air driller (Brassler, USA) at the left anterior junction of the coronal and sagittal sutures to expose the underlying dura.

The piglet was moved onto a controlled cortical impact (CCI) device (University of Georgia Instrument Design and Fabrication Shop; Athens, GA) as previously described (Baker et al., 2019; Kinder et al., 2019). A 15 mm impactor tip was positioned over the intact dura to induce injury with the following parameters: 4m/s velocity, 9mm depth of depression, and 400ms dwell time. Immediately following CCI, the surgical site was flushed with sterile saline and re-apposed with surgical suture without replanting the bone flap. After surgery, piglets were maintained on oxygen until recovered and then placed back in their home pens once ambulatory. For post-operative analgesia, piglets were administered Flunixin (2.2 mg/kg) 12 hours post-surgery and then once daily for an additional 4 days. Oxytetracycline (19.8 mg/kg) was administered as an antibiotic for 5 days post TBI. Piglets were monitored daily for health or signs of abnormal neurological behaviors.

To assess changes over time after TBI, piglets were separated into 4 groups based on sacrifice date: 1 day ($n = 4$), 1 week ($n = 4$), 4 weeks ($n = 4$), and 12 weeks ($n = 6$) post-TBI.

4.2.2 Magnetic Resonance Imaging

Magnetic resonance imaging (MRI) was performed 24 hours (24hrs) and 12 weeks (12wks) post-TBI on a 3.0 Tesla whole-body MR scanner (Siemens MAGNETOM TIM/Trio system, Siemens Healthineers, Erlangen, Germany) using a 12-channel phased array head coil. During MRI acquisition, pigs were maintained under anesthesia using the protocol described for CCI induction and positioned in dorsal recumbency. Standard multiplanar MR brain imaging sequences were performed at the sagittal, coronal, and axial planes including T1-weighted, T2-weighted (T2W) fast spin echo (FSE), T2-weighted fluid attenuated inversion recovery (FLAIR), axial diffusion-weighted imaging (DWI), and diffusion tensor imaging (DTI). Due to scanner complications, group sizes were as follows: T2-weighted FSE and FLAIR, $n = 5$ at 24hrs and 12wks post-TBI; DWI and DTI, $n = 6$ at 24hrs post-TBI and $n = 5$ at 12wks post-TBI.

T2W images were used for volumetric assessments such as lesion volume, which was defined by pixels with abnormal (high or low) signal intensity compared to the same anatomic area on the contralateral hemisphere. To calculate lesion volume, OsiriX software was used to manually define the lesion on a slice by slice basis. The area of each slice was multiplied by the slice thickness (5mm) to obtain the lesion volume of each slice. The slice areas were then summed to obtain total lesion volume. T2W images were also used to measure midline shift which is defined as the maximal horizontal displacement of the septum pellucidum in relation to the midline.(Jacobs et al., 2011; Sauvigny et al., 2016) The distance from the septum pellucidum to the outer border of the cortex was measured for each hemisphere. The midline shift was calculated using the formula: midline shift = (total diameter/2) – contralateral diameter.

DWI sequences were used to generate apparent diffusion coefficient (ADC) maps to measure changes in diffusivity. Mean ADC values were calculated with NIH ImageJ software at a manually drawn region of interest (ROI) that included both the lesion core and perilesion area. The ROI was defined by pixels

with abnormal (high or low) signal intensity compared to the same anatomic area on the contralateral hemisphere. ROIs drawn in a symmetrical site in the contralateral hemisphere served as an internal control. Mean ADC values were obtained by calculating the average signal intensity across all slices. Normalized ADC value was calculated by dividing the ipsilateral ADC value by the contralateral ADC value. All ADC values are reported as 10^{-3} mm^2/s.

DTI sequences were used to generate fractional anisotropy (FA) maps to measure changes in white matter integrity. ROIs were drawn at the internal capsule (IC) and corpus callosum (CC) on one representative slice per animal using NIH ImageJ software at the ipsilateral and contralateral side of the injury (Schneider et al., 2012). Normalized FA value was calculated by dividing the ipsilateral FA value by the contralateral FA value for both the IC and CC.

4.2.3 Gait Analysis

Pigs underwent gait assessment at the following timepoints: 24hrs ($n = 18$) and 12wks post-TBI ($n = 6$). Gait was collected using a GAITFour electronic, pressure-sensitive mat (CIR Systems Inc., Franklin, NJ) that is 7.01 m in length and 0.85 m in width with an active area that is 6.10 m in length and 0.61 m in width. In this arrangement, the active area is a grid, 48 sensors wide by 480 sensors long, totaling $23,040$ sensors. Prior to gait collection, pigs were trained to travel back and forth across the gait mat at a self-selected, consistent two-beat gait using food as a positive reward. During gait collection, pigs were assessed until 4-5 consistent trials were obtained, or up to a maximum of 20 minutes per pig on each testing day. For each trial, a minimum of 3 gait cycles with less than 10% variability in velocity were selected, and all gait parameters were calculated automatically using the GAITFour software. Age-matched, normal pigs ($n = 6$) underwent gait collection at identical time points to TBI pigs in order to account for changes in gait that are attributed to normal growth. The gait parameters assessed are described in Table 4.1. The following parameters are reported as an average of the left front, right front, left hind, and right hind limbs: step length, stride length, percent swing, percent stance, number of sensors, scaled pressures, mean pressure, hind reach, total pressure index, and step/stride ratio.

4.2.4 Statistical Analysis

PCA was implemented to recognize the parameters that explained the most variability of MRI data and gait data at 24hrs and 12wks. The first two principal components (PCs) are considered to be the most important parameters among all the PCs as they explain the most variability compared to other parameters. Simple linear regression was utilized to determine whether MRI parameters are able to predict gait parameters with respect to functional changes. Parameters whose p-values were ≤ 0.05 were considered to be statistically significant.

Table 4.1: Gait Parameters Definition.

Gait Parameter	Definition
Step Length	Distance between corresponding successive points of heel contact of opposing limbs (i.e. right front and left front, right hind and left hind); expressed in cm
Stride Length	Distance between successive points of heel contact of the same hoof (i.e. left front and left front); expressed in cm
Hind Reach	Distance from the heel center of the hind limb to the heel center of the previous front limb on the same side (i.e. left hind to left front); expressed in cm
Step/Stride Ratio	The ratio between step and stride lengths of the same limb
Number of Sensors	The number of sensors activated by contact of each limb
Velocity	Stride Length divided by stride time, expressed in cm/sec
Cadence	Frequency of steps/min during a trial
Percent Stance	Percentage of time during which a limb is in stance phase during one stride cycle (stance time/cycle time)
Percent Swing	Percentage of time during which a limb is in swing phase during one stride cycle (swing time/cycle time)
Mean Pressure	Average pressure of all sensors for one limb
Total Scaled Pressure	The sum of peak pressure values recorded from each activated sensor by a limb during mat contact, represented by the switching levels and reported as a scaled pressure from 0-7 for each sensor
Total Pressure Index (TPI)	Percent distribution of weight across all four limbs. Pigs typically carry 30% of their weight in each front limb and 20% of their weight in each hind limb

4.3 Results

4.3.1 Lesion volume and midline shift explain most of the MRI data variability at 24 hours and 12 weeks post-TBI

Representative T2W images 24hrs (Figure 4.1 A) and 12wks (Figure 4.1 B) post-TBI showed distinct brain lesions. At 24hrs post-TBI, the midline shifted away from the ipsilateral hemisphere (Figure 4.1 C, red line) at the level of the septum pellucidum (yellow line), while at 12wks post-TBI the midline shifted toward the ipsilateral hemisphere (Figure 4.1 D, blue line). These changes in midline shift are indicative of swelling and atrophy at each time point, respectively. At 24hrs and 12wks post-TBI, PCA of lesion volume and midline shift from T2W sequences, normalized ADC maps from DWI sequences, and normalized IC and CC FA values from DTI sequences was performed. The Scree Plot of the five PCs of MRI data at 24hrs post-TBI using covariance criteria showed that PC1 and PC2 explained most of the variability (63.9% and 36%, respectively, Figure 4.1 E). Both PC1 and PC2 are linear combinations of lesion volume and midline shift (Figure 4.1 E). The Biplot of PC1 against PC2 indicated that lesion volume and midline shift are dominant in explaining most of the variability at 24hrs post-TBI (Figure 4.1 F). The Scree Plot of the five PCs of MRI data at 12wks post-TBI using covariance criteria showed that PC1 and PC2 explained most of the variability (69.6% and 29.3%, respectively, Fig 1G). Both PC1 and PC2 are linear combinations of lesion volume and midline shift (Figure 4.1 G). The Biplot of PC1 against PC2 indicated that lesion volume and midline shift are dominant in explaining most of the variability at 12wks post-TBI (Figure 4.1 H). Taken together, these results suggest that lesion volume and midline shift are key MRI indicators of injury severity after TBI.

4.3.2 Velocity, cadence, and stride length explain most of the gait data variability at 24hrs and 12wks post-TBI

Pigs were subjected to gait analysis using a GAITFour electronic, pressure-sensitive mat at 24hrs and 12wks post-TBI (Figure 4.2 A). Activated sensors on the mat were capable of detecting key gait pressure and spatiotemporal parameter changes of the left front (blue), left hind (green), right front (red), and right hind (black) limbs (Figure 4.2 B, black line indicates stride length for the left front limb). The Scree Plot of the first 10 PCs of gait data at 24hrs post-TBI using covariance criteria showed that PC1 and PC2 explained most of the variability (85.5% and 10.9%, respectively, Figure 4.2 C). Only the first 10 PCs are displayed because the percentage of explained variances of the last 6 PCs (PC7-PC12) are all 0%. PC1 is a linear combination of velocity, cadence, and stride length, while PC2 is a linear combination of velocity, cadence, step length, stride length, percent swing, percent stance, and total scaled pressure (Figure 4.2 C). The Biplot of PC1 against PC2 indicated that velocity, cadence, and stride length are dominant in explaining most of the variability at 24hrs post-TBI (Figure 4.2 D). The Scree Plot of the first 10 PCs of gait data at 12wks post-TBI using covariance criteria showed that PC1 and PC2 explained most of the variability (91.1% and 4.8%, respectively, Figure 4.2 E). Similarly, only the first 10 PCs are displayed because the

Figure 4.1: MRI Images and PCA Plots.

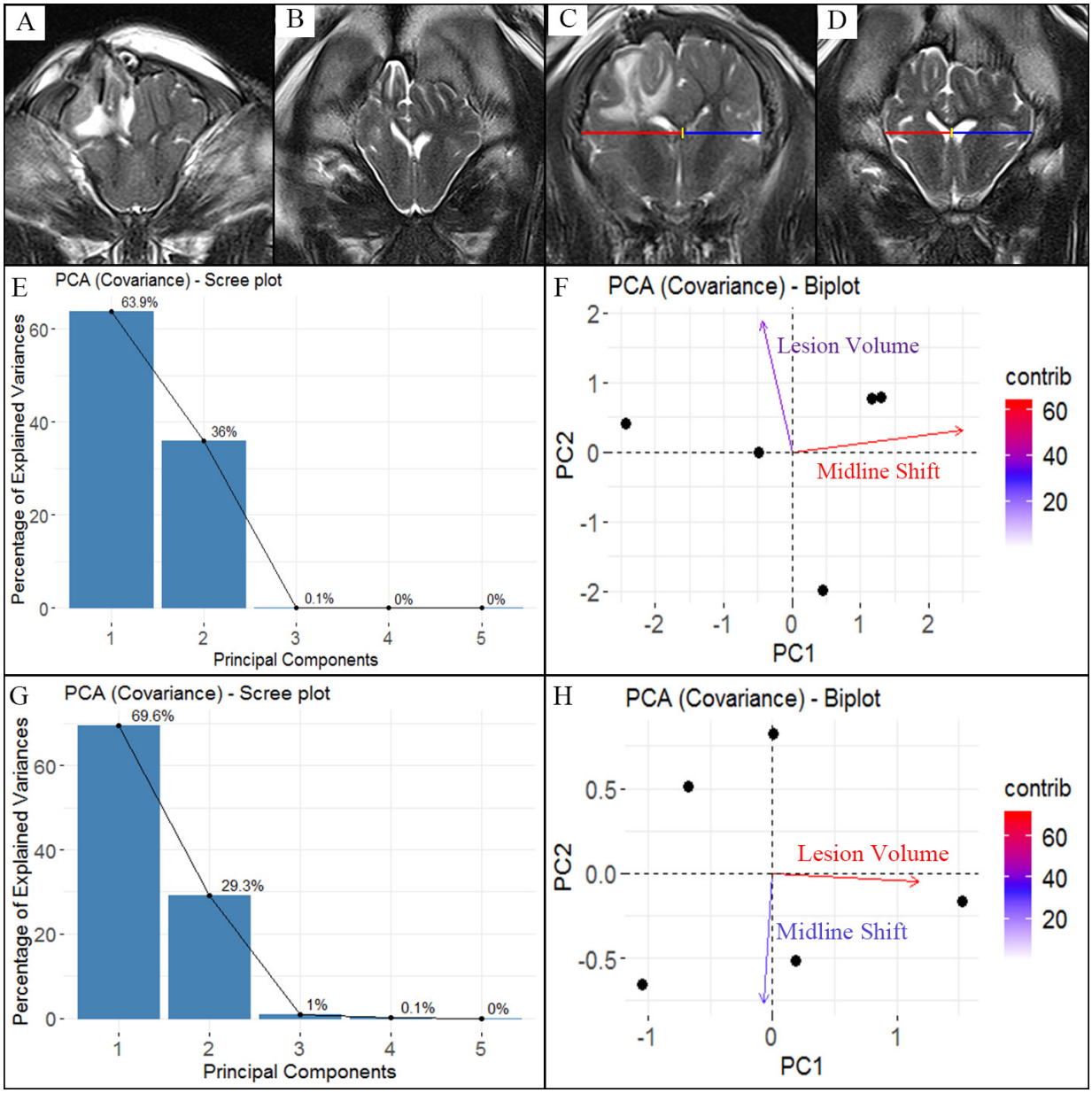


Table 4.2: Linear Model Results.

Model of Parameters	P-value
Stride Length = 96.14 – 8.7*Lesion Volume	0.0306
Step Length = 48.68 – 4.52*Lesion Volume	0.0219
Stride Length = 68.79 – 4.45*Midline Shift	0.0341
Step Length = 34.5 – 2.26*Midline Shift	0.0305

percentage of explained variances of the last 6 PCs (PC7-PC12) are all 0%. PC1 is a linear combination of velocity, cadence, stride length, percent swing, and percent stance, while PC2 is a linear combination of velocity, cadence, step length, stride length, total scaled pressure, and hind reach (Figure 4.2 E). The Biplot of PC1 against PC2 indicated that velocity, cadence, and stride length are dominant in explaining most of the variability at 12wks post-TBI (Figure 4.2 F). Taken together, these results suggest that velocity, cadence, and stride length are key gait indicators of motor function impairments after TBI.

4.3.3 Lesion volume and midline shift predict deficits in stride and step length

Linear regression analysis was performed to determine if MRI parameters predict changes in gait. Combined 24hrs and 12wk post-TBI data showed that increases in lesion volume resulted in a direct decrease in stride length (Figure 4.3 A) and step length (Figure 4.3 B). Lesion volume was found to be significant in predicting stride length (p=0.03) and step length ($p = 0.02$, Table 4.2). Similarly, an increase in midline shift resulted in a direct decrease in stride length (Figure 4.3 C) and step length (Figure 4.3 D). Midline shift was found to be significant in predicting stride length (p=0.03) and step length (p=0.03, Table 4.2). Therefore, these results suggest that lesion size and midline shift can be used as key indications of functional deficits in a pig TBI model.

4.4 Discussion

Multiparametric neuroimaging is becoming increasingly more important for identifying and diagnosing TBI outcomes (Chastain et al., 2009; Sigmund et al., 2007). However, the identification of key MRI biomarkers with potent prognostic value has proven to be challenging given the heterogeneous nature of TBI and the diversity of the patient population. Multivariate projection models may provide a mathematical framework to identify linear combinations of MRI biomarkers with acceptable sensitivity and specificity with regard to predicting outcome (Irimia et al., 2012). In this study, we performed PCA in order to explore the largest sources of variation within MRI datasets and identify MRI parameters that closely co-vary. We found that linear combinations of lesion size and midline shift explain most of the variability of the data at both 24hrs and 12wks post-TBI. In addition, PCA was utilized to explore the largest sources of variation within gait datasets and identify gait parameters that closely co-vary. Linear

Figure 4.2: Gait Images and PCA Plots.

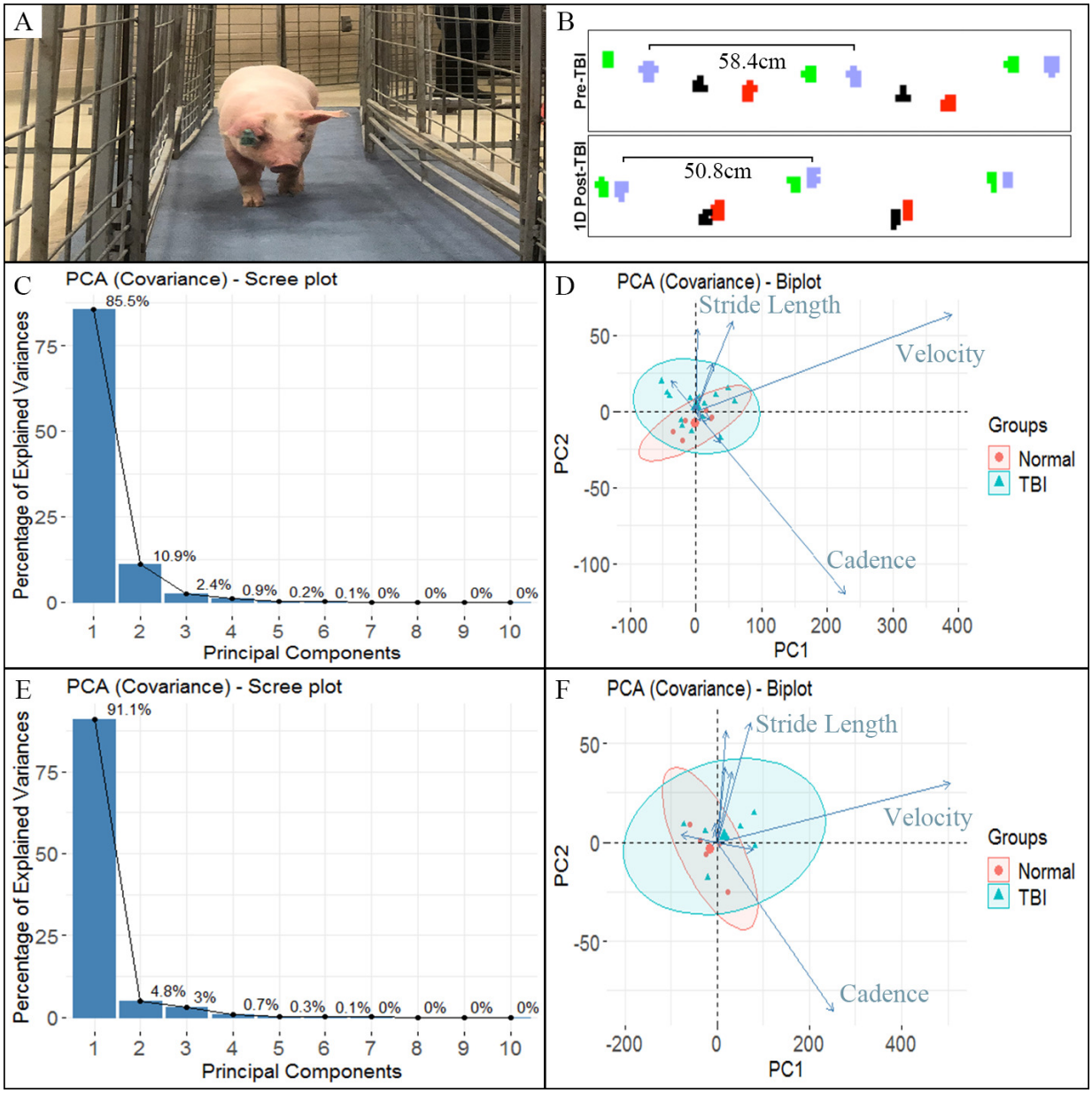
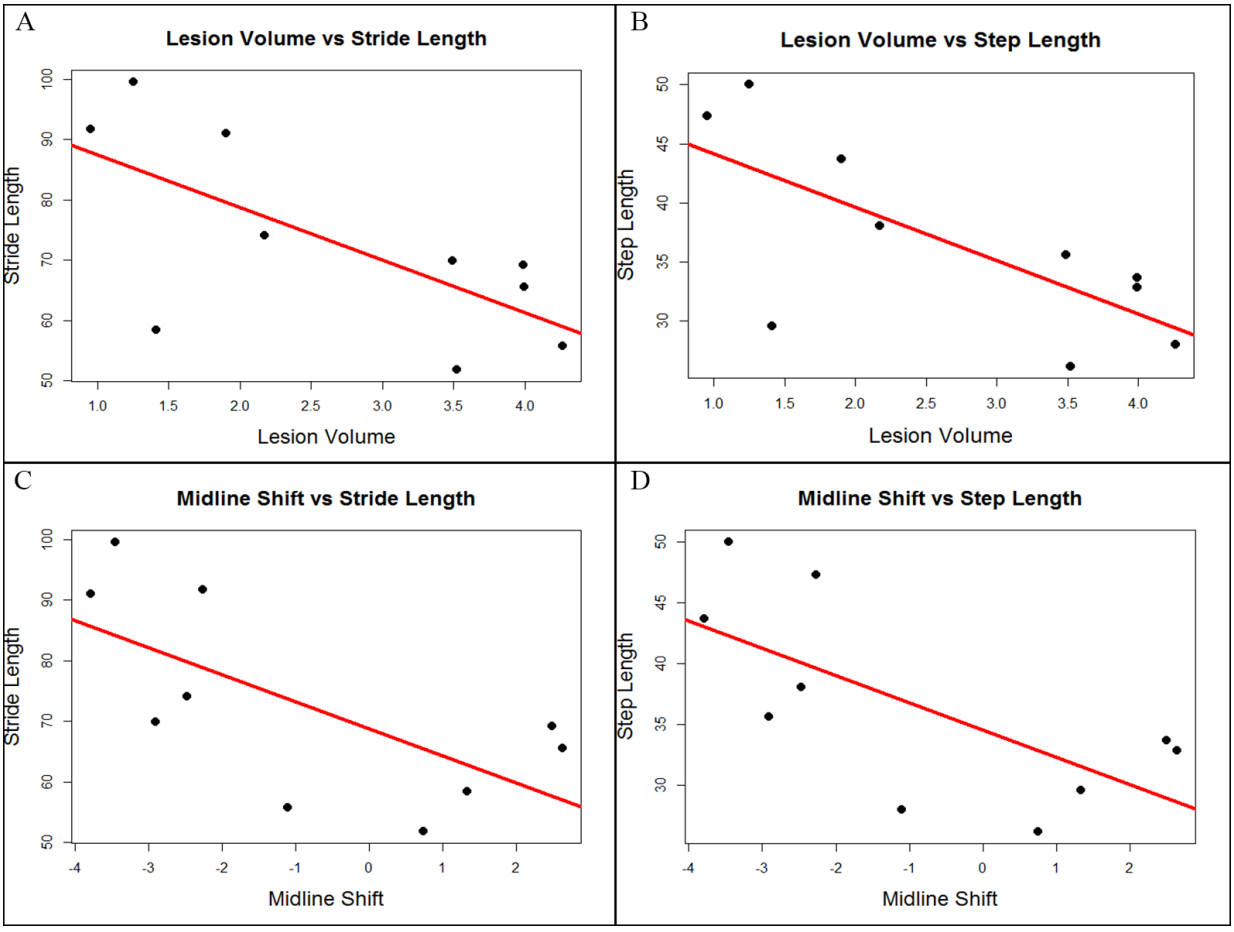


Figure 4.3: Linear Regression Plots.



combinations of velocity, cadence, and stride length were found to explain most of the gait data variability at 24hrs and 12wks post-TBI. Linear regression of MRI and gait parameters combined at 24hrs and 12wks post-TBI revealed both lesion size and midline shift were negatively correlated with stride length and step length. Taken together, in this study we showed that advanced statistical approaches can be used to identify key, clinically relevant MRI biomarkers and predict functional outcomes using MRI metrics in a large animal piglet TBI model that closely recapitulates human TBI.

Injury responses following TBI are complex and heterogeneous which creates a challenging landscape for the development of effective diagnostic and prognostic tools (Allen, 2016; Ziebell & Morganti-Kossmann, 2010). Therefore, the identification and use of potent TBI biomarkers may aid in early diagnosis, guide treatment selections, and provide critical insight of long-term prognosis (Martinez & Stabenfeldt, 2019). A multitude of TBI biomarkers that reflect injury severity and predict outcome are currently being explored such as serum markers (Berger et al., 2005; Kilbaugh et al., 2015), neuroimaging metrics (Sener et al., 2016; Strangman et al., 2010), and functional markers (SIGURDARDOTTIR et al., 2009). The use of machine learning and multivariate statistical models may be well suited to identify potential TBI biomarkers by analyzing large data sets to reveal prospective markers of interest (Irimia et al., 2012; Mateos-Pérez et al., 2018). For example, PCA is an unsupervised dimension reduction technique that generates linear combinations of the original variables by exploring sources of variation within the data set. The first principal component (PC) explains the greatest sources of variation and the subsequent PCs explain the greatest sources of variation beyond the first PC (Irimia et al., 2012). In this study, we found that both PC1 and PC2 are linear combinations of lesion volume and midline shift at 24hrs and 12wks post-TBI which suggests that lesion volume and midline shift are key contributors to injury severity at both acute and chronic timepoints in this model. MRI assessments of lesion volume using conventional approaches such as T2W and FLAIR have been found to correspond to injury severity in both pre-clinical models and human TBI patients (Immonen et al., 2009; Sigmund et al., 2007). Increased lesion volume has been associated with poorer Glasgow Outcome Scale (GOS) scores in adult TBI patients and Glasgow Outcome Scale- Extended Pediatrics (GOS-E Peds) scores in pediatric TBI patients (Chastain et al., 2009; Smitherman et al., 2016). Similarly, increased degree of midline shift in patients with head injuries by CT scan corresponded to injury severity, and a midline shift > 5mm was predictive of reduced favorable outcomes and lower quality of life (Chiewvit et al., 2010; Puffer et al., 2018; Swanson et al., 2012). However, predictions of functional outcomes may be enhanced by using multivariate statistical classification methods that identify linear combinations of MRI metrics. The results from this study provide an important first step in identifying relevant MRI biomarkers of injury severity in a piglet model of traumatic brain injury that can be used in future studies to predict a broad range of functional outcomes such as neurologic function, behavior, cognition and motor function.

Neural damage following TBI can lead to the development of balance instability, decreased motor control, and gait abnormalities (Caeyenberghs et al., 2009; Katz-Leurer et al., 2009a, 2009b). These impairments can lead to changes in a multitude of spatiotemporal gait parameters such as velocity, cadence, step length, stride length, stance percent, and swing percent (G. Williams et al., 2010; G. Williams et al., 2009). Currently, it is not well studied if a single parameter, or combination of parameters, might be

more sensitive to injury and thus more reflective of TBI outcomes. In this study, we performed PCA in order to identify the spatiotemporal gait parameters that contributed to the largest sources of variation in our gait data sets. We found that velocity, cadence, and stride length were dominant in explaining most of the variability at 24hrs and 12wks post-TBI. A number of clinical studies have observed significant reductions in velocity and cadence following TBI that can persist for months to even years in both adult and pediatric populations (Beretta et al., 2009; Kuhtz-Buschbeck et al., 2003; Martini et al., 2011; G. Williams et al., 2009). Consequently, as a result of a decreased velocity and motor stability, stride length may also become reduced in TBI patients (Chou et al., 2004; Kuhtz-Buschbeck et al., 2003). Studies that employ multivariate statistical approaches that consider several gait parameters at the same time are limited. In a recent study, Pauk, J. et al. performed bicluster analysis to identify groups of patients with similar spatiotemporal gait patterns in different joints. However, the goal of bicluster analysis in this case was to identify specific patients rather than specific gait parameters that reflect TBI outcomes (Pauk & Minta-Bielecka, 2016). To our knowledge, this is the first study in a piglet TBI model to provide important information regarding potential gait biomarkers for injury using PCA. These results can be used in future studies to better predict TBI outcomes.

TBI severity has been implicated to play an important role in motor function outcomes, especially in children who sustain a TBI at a young age (Gagnon et al., 2004). Reports indicate that the more severe the injury, the less likely a child is to achieve full recovery of physical function (Dumas & Carey, 2002; Grados et al., 2001). Increasing lesion size is well established to reflect TBI severity (Baker et al., 2019; Washington et al., 2012). In this study, linear regression was performed to examine the relationship between lesion size and different spatiotemporal gait parameters. We found that increasing lesion size is predictive of decreases in stride and step length in this model. Similarly, increasing injury severity and lesion size have been found to be associated with motor function impairments in other pre-clinical rodent (Beaumont et al., 1999; Tsenter et al., 2008) and pig (Baker et al., 2019) TBI models. In pediatric TBI patients, increases in TBI severity were found to have a negative impact on motor performance and clinical measures (Jaffe et al., 1993; Kuhtz-Buschbeck et al., 2003). Though direct investigations of the relationship between TBI lesion size and changes in spatiotemporal gait parameters are lacking in TBI patient populations, brain lesion size and location have been found to correlate with motor and functional outcomes in ischemic stroke patients (C.-L. Chen et al., 2000). This is especially true in cases where ischemic lesions involve significant portions of the corticospinal tract, which is highly involved in regulating motor activity (Feng et al., 2015; Zhu et al., 2010). In addition, linear regression analysis in this study revealed that increases in midline shift are also predictive of decreases in step and stride length in this model. In human TBI patients, the presence of a midline shift >5mm was found to be associated with greater need of assistance with ambulation and worse GOS-E outcomes, but no direct comparisons to spatiotemporal gait parameters have been investigated (Englander et al., 2003; Puffer et al., 2018). The results from this study have identified an exciting and clinically translatable predictive relationship between the MRI parameters lesion size and midline shift and the gait spatiotemporal parameters stride length and step length. The use of this approach may aid in more precise predictions of motor impairments following TBI.

4.5 Conclusion

In this study we have utilized advanced statistical approaches to identify potential MRI and functional biomarkers in a piglet TBI model. PCA revealed that the MRI parameters lesion volume and midline shift and the gait parameters velocity, cadence, and stride length may serve as potential biomarkers that are most reflective of TBI outcomes. Lesion volume and midline shift were also found to be significantly correlated with changes in stride and step length which lends support to the clinical utilization of MRI biomarkers to predict motor function outcomes. In the future, this model can be used to further explore the prognostic value of MRI biomarkers in predicting other functional outcomes such as behavior and cognition.

4.6 Research Outcomes

In this research, I learned different bioscience knowledge and concepts as well as I gained collaboration experiences, which is very helpful for both study and work. This was my first time learned and used Photoshop to polish figures and graphs, and it was also my first learned how to use Endnote to manage references. We published this work to the Neural Regeneration Research journal.

Chapter 5

Conclusion

Several years of study for my doctoral degree at the University of Georgia has been one of the most beautiful experience and precious memory in my life. I am extremely grateful to the university as well as to what were offered to me (assistantship, fund and grants, and more importantly, opportunities to learn).

I started with taking different statistics courses and I learned a lot of knowledge from different professors. After the completion of all required course, I began my research under the guidance of my advisor, Dr. Abhyuday Mandal.

During years of my research, I gained knowledge on design of experiments including theories, algorithms, and applications. My first research topic is to study a new algorithm that could be potentially used for solving complex design problems. After that, I researched the area of optimal computer experiments which opened the door of a new world to me. Later, based on my cumulative research knowledge, I invented a brand new algorithm that could identify different types of designs, and I had the opportunity to apply my statistical knowledge to a real world application that could be helpful for the identification of key parameters in the diagnosis of traumatic brain injury.

In the future, I hope to continually enhance my knowledge in my research area and make more scientific breakthroughs.

Bibliography

Allen, K. A. (2016). Pathophysiology and treatment of severe traumatic brain injuries in children. *The Journal of neuroscience nursing: journal of the American Association of Neuroscience Nurses*, *48*(1), 15.

Antony, J. (2014). *Design of experiments for engineers and scientists*. Elsevier.

Ba, S. (2015). *Slhd: Maximin-distance (sliced) latin hypercube designs* [R package version 2.1-1]. https://CRAN.R-project.org/package=SLHD

Ba, S., & Joseph, V. R. (2018). *Maxpro: Maximum projection designs* [R package version 4.1-2]. https://CRAN.R-project.org/package=MaxPro

Ba, S., Myers, W. R., & Brenneman, W. A. (2015). Optimal sliced Latin hypercube designs. *Technometrics*, *57*(4), 479–487.

Baker, E. W., Kinder, H. A., Hutcheson, J. M., Duberstein, K. J. J., Platt, S. R., Howerth, E. W., & West, F. D. (2019). Controlled cortical impact severity results in graded cellular, tissue, and functional responses in a piglet traumatic brain injury model. *Journal of neurotrauma*, *36*(1), 61–73.

Basturk, B. (2006). An artificial bee colony (abc) algorithm for numeric function optimization. *IEEE Swarm Intelligence Symposium, Indianapolis, IN, USA, 2006*.

Beaumont, A., Marmarou, A., Czigner, A., Yamamoto, M., Demetriadou, K., Shirotani, T., Marmarou, C., & Dunbar, J. (1999). The impact-acceleration model of head injury: Injury severity predicts motor and cognitive performance after trauma. *Neurological research*, *21*(8), 742–754.

Beretta, E., Cimolin, V., Piccinini, L., Carla Turconi, A., Galbiati, S., Crivellini, M., Galli, M., & Strazzer, S. (2009). Assessment of gait recovery in children after traumatic brain injury. *Brain injury*, *23*(9), 751–759.

Berger, R. P., Adelson, P. D., Pierce, M. C., Dulani, T., Cassidy, L. D., & Kochanek, P. M. (2005). Serum neuron-specific enolase, s100b, and myelin basic protein concentrations after inflicted and noninflicted traumatic brain injury in children. *Journal of Neurosurgery: Pediatrics*, *103*(1), 61–68.

Borkowski, J. J. (2003). Using a genetic algorithm to generate small exact response surface designs. *Journal of Probability and Statistical Science*, *1*(1), 65–88.

Butler, N. A. (2001). Optimal and orthogonal Latin hypercube designs for computer experiments. *Biometrika*, *88*(3), 847–857.

Caeyenberghs, K., Leemans, A., Geurts, M., Taymans, T., Linden, C. V., Smits-Engelsman, B. C., Sunaert, S., & Swinnen, S. P. (2010). Brain-behavior relationships in young traumatic brain injury patients: Dti metrics are highly correlated with postural control. *Human brain mapping*, *31*(7), 992–1002.

Caeyenberghs, K., Wenderoth, N., Smits-Engelsman, B., Sunaert, S., & Swinnen, S. (2009). Neural correlates of motor dysfunction in children with traumatic brain injury: Exploration of compensatory recruitment patterns. *Brain*, *132*(3), 684–694.

Chapman, W. L., Welch, W. J., Bowman, K. P., Sacks, J., & Walsh, J. E. (1994). Arctic sea ice variability: Model sensitivities and a multidecadal simulation. *Journal of Geophysical Research: Oceans*, *99*(C1), 919–935.

Chastain, C. A., Oyoyo, U. E., Zipperman, M., Joo, E., Ashwal, S., Shutter, L. A., & Tong, K. A. (2009). Predicting outcomes of traumatic brain injury by imaging modality and injury distribution. *Journal of neurotrauma*, *26*(8), 1183–1196.

Chen, C.-L., Tang, F.-T., Chen, H.-C., Chung, C.-Y., & Wong, M.-K. (2000). Brain lesion size and location: Effects on motor recovery and functional outcome in stroke patients. *Archives of physical medicine and rehabilitation*, *81*(4), 447–452.

Chen, R.-B., Chang, S.-P., Wang, W., Tung, H.-C., & Wong, W. K. (2015). Minimax optimal designs via particle swarm optimization methods. *Statistics and Computing*, *25*(5), 975–988.

Chen, R.-B., Hsieh, D.-N., Hung, Y., & Wang, W. (2013). Optimizing Latin hypercube designs by particle swarm. *Statistics and computing*, *23*(5), 663–676.

Chiewvit, P., Tritakarn, S.-o., Nanta-aree, S., & Suthipongchai, S. (2010). Degree of midline shift from ct scan predicted outcome in patients with head injuries. *Medical journal of the Medical Association of Thailand*, *93*(1), 99.

Chou, L.-S., Kaufman, K. R., Walker-Rabatin, A. E., Brey, R. H., & Basford, J. R. (2004). Dynamic instability during obstacle crossing following traumatic brain injury. *Gait & posture*, *20*(3), 245–254.

Cioppa, T. M., & Lucas, T. W. (2007). Efficient nearly orthogonal and space-filling Latin hypercubes. *Technometrics*, *49*(1), 45–55.

Conrad, M. S., Dilger, R. N., & Johnson, R. W. (2012). Brain growth of the domestic pig (sus scrofa) from 2 to 24 weeks of age: A longitudinal mri study. *Developmental neuroscience*, *34*(4), 291–298.

Conrad, M. S., & Johnson, R. W. (2015). The domestic piglet: An important model for investigating the neurodevelopmental consequences of early life insults. *Annu. Rev. Anim. Biosci.*, *3*(1), 245–264.

Cooper, J. M., Catroppa, C., Beauchamp, M. H., Eren, S., Godfrey, C., Ditchfield, M., & Anderson, V. A. (2014). Attentional control ten years post-childhood traumatic brain injury: The impact of lesion presence, location, and severity in adolescence and early adulthood. *Journal of neurotrauma*, *31*(8), 713–721.

Cornell, J. A. (1973). Experiments with mixtures: A review. *Technometrics*, *15*(3), 437–455.

Costine, B. A., Missios, S., Taylor, S. R., McGuone, D., Smith, C. M., Dodge, C. P., Harris, B. T., & Duhaime, A.-C. (2015). The subventricular zone in the immature piglet brain: Anatomy and exodus of neuroblasts into white matter after traumatic brain injury. *Developmental neuroscience*, *37*(2), 115–130.

Dean, A., Voss, D., Draguljić, D., et al. (2017). *Design and analysis of experiments*. Springer International Publishing.

Dobbing, J., & Sands, J. (1979). Comparative aspects of the brain growth spurt. *Early human development*, *3*(1), 79–83.

Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, *1*(4), 28–39.

Drijkoningen, D., Chalavi, S., Sunaert, S., Duysens, J., Swinnen, S. P., & Caeyenberghs, K. (2017). Regional gray matter volume loss is associated with gait impairments in young brain-injured individuals. *Journal of neurotrauma*, *34*(5), 1022–1034.

Duhaime, A.-C., Hunter, J. V., Grate, L. L., Kim, A., Golden, J., Demidenko, E., & Harris, C. (2003). Magnetic resonance imaging studies of age-dependent responses to scaled focal brain injury in the piglet. *Journal of neurosurgery*, *99*(3), 542–548.

Dumas, H. M., & Carey, T. (2002). Motor skill and mobility recovery outcomes of children and youth with traumatic brain injury. *Physical & Occupational Therapy in Pediatrics*, *22*(3-4), 73–99.

Emich, F. (1900). Über explosive gasgemenge. *Monatshefte für Chemie und verwandte Teile anderer Wissenschaften*, *21*(10), 1061–1078.

Englander, J., Cifu, D. X., Wright, J. M., & Black, K. (2003). The association of early computed tomography scan findings and ambulation, self-care, and supervision needs at rehabilitation discharge and at 1 year after traumatic brain injury. *Archives of Physical Medicine and Rehabilitation*, *84*(2), 214–220.

Fang, K.-T., Li, R., & Sudjianto, A. (2005). *Design and modeling for computer experiments*. CRC press.

Fang, K.-T., Ma, C.-X., & Winker, P. (2002). Centered L_2-discrepancy of random sampling and Latin hypercube design, and construction of uniform designs. *Mathematics of Computation*, *71*(237), 275–296.

Faul, M., Wald, M. M., Xu, L., & Coronado, V. G. (2010). Traumatic brain injury in the united states: Emergency department visits, hospitalizations, and deaths, 2002-2006.

Fedorov, V. V. (2013). *Theory of optimal experiments*. Elsevier.

Feng, W., Wang, J., Chhatbar, P. Y., Doughty, C., Landsittel, D., Lioutas, V.-A., Kautz, S. A., & Schlaug, G. (2015). Corticospinal tract lesion load: An imaging biomarker for stroke motor outcomes. *Annals of neurology*, *78*(6), 860–870.

Flynn, T. J. (1984). Developmental changes of myelin-related lipids in brain of miniature swine. *Neurochemical Research*, *9*(7), 935–945.

Furlanetto, S., Cirri, M., Piepel, G., Mennini, N., & Mura, P. (2011). Mixture experiment methods in the development and optimization of microemulsion formulations. *Journal of pharmaceutical and biomedical analysis*, *55*(4), 610–617.

Gagnon, I., Swaine, B., Friedman, D., & Forget, R. (2004). Children show decreased dynamic balance after mild traumatic brain injury. *Archives of physical medicine and rehabilitation*, *85*(3), 444–452.

Galloway, N. R., Tong, K. A., Ashwal, S., Oyoyo, U., & Obenaus, A. (2008). Diffusion-weighted imaging improves outcome prediction in pediatric traumatic brain injury. *Journal of neurotrauma*, *25*(10), 1153–1162.

Garcia, F. J. M., & Pérez, J. A. M. (2008). Jumping frogs optimization: A new swarm method for discrete optimization. *Documentos de Trabajo del DEIOC*, *3*.

Georgiou, S. D. (2009). Orthogonal Latin hypercube designs from generalized orthogonal designs. *Journal of Statistical Planning and Inference*, *139*(4), 1530–1540.

Georgiou, S. D., & Efthimiou, I. (2014). Some classes of orthogonal Latin hypercube designs. *Statistica Sinica*, *24*(1), 101–120.

Gieling, E. T., Schuurman, T., Nordquist, R. E., & Staay, F. (2011). The pig as a model animal for studying cognition and neurobehavioral disorders. *Molecular and functional models in neuropsychiatry*, 359–383.

Goldberg, D. E. (1989). Genetic algorithms in search. *Optimization, and MachineLearning*.

Grados, M., Slomine, B., Gerring, J., Vasa, R., Bryan, N., & Denckla, M. (2001). Depth of lesion model in children and adolescents with moderate to severe traumatic brain injury: Use of spgr mri to predict severity and outcome. *Journal of Neurology, Neurosurgery & Psychiatry*, *70*(3), 350–358.

Gramacy, R. B. (2020). *Surrogates: Gaussian process modeling, design and optimization for the applied sciences* [http://bobby.gramacy.com/surrogates/]. Chapman Hall/CRC.

Grate, L. L., Golden, J. A., Hoopes, P. J., Hunter, J. V., & Duhaime, A.-C. (2003). Traumatic brain injury in piglets of different ages: Techniques for lesion analysis using histology and magnetic resonance imaging. *Journal of neuroscience methods*, *123*(2), 201–206.

Grosso, A., Jamali, A., & Locatelli, M. (2009). Finding maximin Latin hypercube designs by iterated local search heuristics. *European Journal of Operational Research*, *197*(2), 541–547.

Guild, E. B., & Levine, B. (2015). Functional correlates of midline brain volume loss in chronic traumatic brain injury. *Journal of the International Neuropsychological Society*, *21*(8), 650–655.

Hansen, P., Mladenović, N., & Pérez, J. A. M. (2010). Variable neighbourhood search: Methods and applications. *Annals of Operations Research*, *175*(1), 367–407.

Harari, O., Bingham, D., Dean, A., & Higdon, D. (2018). Computer experiments: Prediction accuracy, sample size and model complexity revisited. *Statistica Sinica*, 899–919.

Heredia-Langner, A., Carlyle, W. M., Montgomery, D. C., Borror, C. M., & Runger, G. C. (2003). Genetic algorithms for the construction of d-optimal designs. *Journal of quality technology*, *35*(1), 28–46.

Hernandez, L. N., & Nachtsheim, C. J. (2018). Fast computation of exact g-optimal designs via i λ-optimality. *Technometrics*, *60*(3), 297–305.

Hickernell, F. (1998). A generalized discrepancy and quadrature error bound. *Mathematics of computation*, *67*(221), 299–322.

Holland, J. (1975). An efficient genetic algorithm for the traveling salesman problem. *European Journal of Operational Research*, *145*, 606–617.

Holland, J. H. et al. (1992). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Immonen, R. J., Kharatishvili, I., Gröhn, H., Pitkänen, A., & Gröhn, O. H. (2009). Quantitative mri predicts long-term structural and functional outcome after experimental traumatic brain injury. *Neuroimage*, *45*(1), 1–9.

Irimia, A., Wang, B., Aylward, S. R., Prastawa, M. W., Pace, D. F., Gerig, G., Hovda, D. A., Kikinis, R., Vespa, P. M., & Van Horn, J. D. (2012). Neuroimaging of structural pathology and connectomics in traumatic brain injury: Toward personalized outcome prediction. *NeuroImage: Clinical*, *1*(1), 1–17.

Jacobs, B., Beems, T., van der Vliet, T. M., Diaz-Arrastia, R. R., Borm, G. F., & Vos, P. E. (2011). Computed tomography and outcome in moderate and severe traumatic brain injury: Hematoma volume and midline shift revisited. *Journal of neurotrauma*, *28*(2), 203–215.

Jaffe, K. M., Fay, G. C., Polissar, N. L., Martin, K. M., Shurtleff, H. A., J'May, B. R., & Winn, H. R. (1993). Severity of pediatric traumatic brain injury and neurobehavioral recovery at one year—a cohort study. *Archives of Physical Medicine and Rehabilitation*, *74*(6), 587–595.

Jain, K. K. (2019). Neuroprotection in traumatic brain injury. *The Handbook of Neuroprotection*, 281–336.

Jin, R., Chen, W., & Sudjianto, A. (2005). An efficient algorithm for constructing optimal design of computer experiments. *Journal of statistical planning and inference*, *134*(1), 268–287.

Johnson, M. E., Moore, L. M., & Ylvisaker, D. (1990). Minimax and maximin distance designs. *Journal of statistical planning and inference*, *26*(2), 131–148.

Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, *13*(4), 455–492.

Joseph, V. R., Gul, E., & Ba, S. (2015). Maximum projection designs for computer experiments. *Biometrika*, *102*(2), 371–380.

Joseph, V. R., & Hung, Y. (2008). Orthogonal-maximin Latin hypercube designs. *Statistica Sinica*, 171–186.

Karlsson, M., Pukenas, B., Chawla, S., Ehinger, J. K., Plyler, R., Stolow, M., Gabello, M., Hugerth, M., Elmér, E., Hansson, M. J., et al. (2019). Neuroprotective effects of cyclosporine in a porcine pre-clinical trial of focal traumatic brain injury. *Journal of Neurotrauma*, *36*(1), 14–24.

Katz-Leurer, M., Rotem, H., Keren, O., & Meyer, S. (2009a). Balance abilities and gait characteristics in post-traumatic brain injury, cerebral palsy and typically developed children. *Developmental neurorehabilitation*, *12*(2), 100–105.

Katz-Leurer, M., Rotem, H., Keren, O., & Meyer, S. (2009b). The relationship between step variability, muscle strength and functional walking performance in children with post-traumatic brain injury. *Gait & Posture*, *29*(1), 154–157.

Kennedy, J. (2006). Swarm intelligence. *Handbook of nature-inspired and innovative computing* (pp. 187–219). Springer.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95-International Conference on Neural Networks*, *4*, 1942–1948.

Kilbaugh, T. J., Lvova, M., Karlsson, M., Zhang, Z., Leipzig, J., Wallace, D. C., & Margulies, S. S. (2015). Peripheral blood mitochondrial dna as a biomarker of cerebral mitochondrial dysfunction following traumatic brain injury in a porcine model. *PLoS One*, *10*(6), e0130927.

Kinder, H. A., Baker, E. W., Wang, S., Fleischer, C. C., Howerth, E. W., Duberstein, K. J., Mao, H., Platt, S. R., & West, F. D. (2019). Traumatic brain injury results in dynamic brain structure changes leading to acute and chronic motor function deficits in a pediatric piglet model. *Journal of Neurotrauma*, *36*(20), 2930–2942.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, *220*(4598), 671–680.

Kuhtz-Buschbeck, J. P., Hoppe, B., Gölge, M., Dreesmann, M., Damm-Stünitz, U., & Ritz, A. (2003). Sensorimotor recovery in children after traumatic brain injury: Analyses of gait, gross motor, and fine motor skills. *Developmental medicine and child neurology*, *45*(12), 821–828.

Leary, S., Bhaskar, A., & Keane, A. (2003). Optimal orthogonal-array-based Latin hypercubes. *Journal of Applied Statistics*, *30*(5), 585–598.

Lee, S.-Y., Kim, S. S., Kim, C.-H., Park, S.-W., Park, J. H., & Yeo, M. (2012). Prediction of outcome after traumatic brain injury using clinical and neuroimaging variables. *Journal of clinical neurology*, *8*(3), 224–229.

Liefvendahl, M., & Stocki, R. (2006). A study on algorithms for optimization of Latin hypercubes. *Journal of statistical planning and inference*, *136*(9), 3231–3247.

Lin, C. D., Anderson-Cook, C. M., Hamada, M. S., Moore, L. M., & Sitter, R. R. (2015). Using genetic algorithms to design experiments: A review. *Quality and Reliability Engineering International*, *31*(2), 155–167.

Lin, C. D., Mukerjee, R., & Tang, B. (2009). Construction of orthogonal and nearly orthogonal Latin hypercubes. *Biometrika*, *96*(1), 243–247.

Lin, D. K. J. (1993). A new class of supersaturated designs. *Technometrics*, *35*(1), 28–31.

Lin, D. K., & Peng, J. (2019). Order-of-addition experiments: A review and some new thoughts. *Quality Engineering*, *31*(1), 49–59.

Loeppky, J. L., Sacks, J., & Welch, W. J. (2009). Choosing the sample size of a computer experiment: A practical guide. *Technometrics*, *51*(4), 366–376.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2019). Iterated local search: Framework and applications. *Handbook of metaheuristics* (pp. 129–168). Springer.

Lukemire, J., Mandal, A., & Wong, W. K. (2019). D-qpso: A quantum-behaved particle swarm technique for finding d-optimal designs with discrete and continuous factors and a binary response. *Technometrics*, *61*(1), 77–87.

Mak, S., Sung, C.-L., Wang, X., Yeh, S.-T., Chang, Y.-H., Joseph, V. R., Yang, V., & Wu, C. J. (2018). An efficient surrogate model for emulation and physics extraction of large eddy simulations. *Journal of the American Statistical Association*, *113*(524), 1443–1456.

Marklund, N. (2016). Rodent models of traumatic brain injury: Methods and challenges. *Injury models of the central nervous system* (pp. 29–46). Springer.

Martinez, B. I., & Stabenfeldt, S. E. (2019). Current trends in biomarker discovery and analysis tools for traumatic brain injury. *Journal of biological engineering*, *13*(1), 1–12.

Martini, D. N., Sabin, M. J., DePesa, S. A., Leal, E. W., Negrete, T. N., Sosnoff, J. J., & Broglio, S. P. (2011). The chronic effects of concussion on gait. *Archives of physical medicine and rehabilitation*, *92*(4), 585–589.

Mateos-Pérez, J. M., Dadar, M., Lacalle-Aurioles, M., Iturria-Medina, Y., Zeighami, Y., & Evans, A. C. (2018). Structural neuroimaging as clinical predictor: A review of machine learning applications. *NeuroImage: Clinical*, *20*, 506–522.

McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, *21*(2), 239–245.

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey wolf optimizer. *Advances in engineering software*, *69*, 46–61.

Morris, M. D., & Mitchell, T. J. (1995). Exploratory designs for computational experiments. *Journal of statistical planning and inference*, *43*(3), 381–402.

Paredes, M. F., James, D., Gil-Perotin, S., Kim, H., Cotter, J. A., Ng, C., Sandoval, K., Rowitch, D. H., Xu, D., McQuillen, P. S., et al. (2016). Extensive migration of young neurons into the infant human frontal lobe. *Science*, *354*(6308), aaf7073.

Pauk, J., & Minta-Bielecka, K. (2016). A new classification of hemiplegia gait patterns based on bicluster analysis of joint moments. *Acta of Bioengineering and Biomechanics*, *18*(4).

Peng, J., Mukerjee, R., & Lin, D. K. J. (2019). Design of order-of-addition experiments. *Biometrika*, *106*(3), 683–694.

Price, K., Storn, R. M., & Lampinen, J. A. (2006). *Differential evolution: A practical approach to global optimization*. Springer Science & Business Media.

Puffer, R. C., Yue, J. K., Mesley, M., Billigen, J. B., Sharpless, J., Fetzick, A. L., Puccio, A., Diaz-Arrastia, R., & Okonkwo, D. O. (2018). Long-term outcome in traumatic brain injury patients with midline shift: A secondary analysis of the phase 3 cobrit clinical trial. *Journal of Neurosurgery*, *131*(2), 596–603.

Pukelsheim, F. (2006). *Optimal design of experiments*. SIAM.

Qian, P. Z. (2012). Sliced Latin hypercube designs. *Journal of the American Statistical Association*, *107*(497), 393–399.

Qiu, J., Chen, R.-B., Wang, W., & Wong, W. K. (2014). Using animal instincts to design efficient biomedical studies via particle swarm optimization. *Swarm and evolutionary computation*, *18*, 1–10.

Rosenthal, G., Morabito, D., Cohen, M., Roeytenberg, A., Derugin, N., Panter, S. S., Knudson, M. M., & Manley, G. (2008). Use of hemoglobin-based oxygen-carrying solution–201 to improve resuscitation parameters and prevent secondary brain injury in a swine model of traumatic brain injury and hemorrhage. *Journal of neurosurgery*, *108*(3), 575–587.

Saatman, K. E., Duhaime, A.-C., Bullock, R., Maas, A. I., Valadka, A., & Manley, G. T. (2008). Classification of traumatic brain injury for targeted therapies. *Journal of neurotrauma*, *25*(7), 719–738.

Sacks, J., Schiller, S. B., & Welch, W. J. (1989). Designs for computer experiments. *Technometrics*, *31*(1), 41–47.

Santner, T. J., Williams, B. J., Notz, W., & Williams, B. J. (2003). *The design and analysis of computer experiments* (Vol. 1). Springer.

Sauvigny, T., Göttsche, J., Vettorazzi, E., Westphal, M., & Regelsberger, J. (2016). New radiologic parameters predict clinical outcome after decompressive craniectomy. *World neurosurgery*, *88*, 519–525.

Schneider, C. A., Rasband, W. S., & Eliceiri, K. W. (2012). Nih image to imagej: 25 years of image analysis. *Nature methods*, *9*(7), 671–675.

Sener, S., Van Hecke, W., Feyen, B. F., Van der Steen, G., Pullens, P., Van de Hauwe, L., Menovsky, T., Parizel, P. M., Jorens, P. G., & Maas, A. I. (2016). Diffusion tensor imaging: A possible biomarker in severe traumatic brain injury and aneurysmal subarachnoid hemorrhage? *Neurosurgery*, *79*(6), 786–793.

Shewry, M. C., & Wynn, H. P. (1987). Maximum entropy sampling. *Journal of applied statistics*, *14*(2), 165–170.

Sigmund, G. A., Tong, K. A., Nickerson, J. P., Wall, C. J., Oyoyo, U., & Ashwal, S. (2007). Multimodality comparison of neuroimaging in pediatric traumatic brain injury. *Pediatric neurology*, *36*(4), 217–226.

SIGURDARDOTTIR, S., ANDELIC, N., ROE, C., & SCHANKE, A.-K. (2009). Cognitive recovery and predictors of functional outcome 1 year after traumatic brain injury. *Journal of the International Neuropsychological Society*, *15*(5), 740–750. https://doi.org/10.1017/S1355617709990452

Simon, D. (2008). Biogeography-based optimization. *IEEE transactions on evolutionary computation*, *12*(6), 702–713.

Smitherman, E., Hernandez, A., Stavinoha, P. L., Huang, R., Kernie, S. G., Diaz-Arrastia, R., & Miles, D. K. (2016). Predicting outcome after pediatric traumatic brain injury by early magnetic resonance imaging lesion location and volume. *Journal of neurotrauma*, *33*(1), 35–48.

Snee, R. D., & Marquardt, D. W. (1976). Screening concepts and designs for experiments with mixtures. *Technometrics*, *18*(1), 19–29.

Stander, P. E. (1992). Cooperative hunting in lions: The role of the individual. *Behavioral ecology and sociobiology*, *29*(6), 445–454.

Steinberg, D. M., & Lin, D. K. J. (2006). A construction method for orthogonal Latin hypercube designs. *Biometrika*, *93*(2), 279–288.

Stokes, Z., Mandal, A., & Wong, W. K. (2020). Using differential evolution to design optimal experiments. *Chemometrics and Intelligent Laboratory Systems*, *199*, 103955.

Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, *11*(4), 341–359.

Strangman, G. E., O'Neil-Pirozzi, T. M., Supelana, C., Goldstein, R., Katz, D. I., & Glenn, M. B. (2010). Regional brain morphometry predicts memory rehabilitation outcome after traumatic brain injury. *Frontiers in human neuroscience*, *4*, 182.

Sun, F., Liu, M.-Q., & Lin, D. K. J. (2009). Construction of orthogonal Latin hypercube designs. *Biometrika*, *96*(4), 971–974.

Sun, F., Liu, M.-Q., & Lin, D. K. J. (2010). Construction of orthogonal Latin hypercube designs with flexible run sizes. *Journal of Statistical Planning and Inference*, *140*(11), 3236–3242.

Sun, F., & Tang, B. (2017). A general rotation method for orthogonal Latin hypercubes. *Biometrika*, *104*(2), 465–472.

Swanson, J. O., Vavilala, M. S., Wang, J., Pruthi, S., Fink, J., Jaffe, K. M., Durbin, D., Koepsell, T., Temkin, N., & Rivara, F. P. (2012). Association of initial ct findings with quality-of-life outcomes for traumatic brain injury in children. *Pediatric radiology*, *42*(8), 974–981.

Tang, B. (1993). Orthogonal array-based Latin hypercubes. *Journal of the American statistical association*, *88*(424), 1392–1397.

Taylor, C. A., Bell, J. M., Breiding, M. J., & Xu, L. (2017). Traumatic brain injury–related emergency department visits, hospitalizations, and deaths—united states, 2007 and 2013. *MMWR Surveillance Summaries*, *66*(9), 1.

Tsenter, J., Beni-Adani, L., Assaf, Y., Alexandrovich, A. G., Trembovler, V., & Shohami, E. (2008). Dynamic changes in the recovery after traumatic brain injury in mice: Effect of injury severity on t2-weighted mri abnormalities, and motor and cognitive functions. *Journal of neurotrauma*, *25*(4), 324–333.

Van Nostrand, R. (1995). Design of experiments where the order of addition is important. *ASA proceedings of the Section on Physical and Engineering Sciences*, 155–160.

Voelkel, J. G. (2019). The design of order-of-addition experiments. *Journal of Quality Technology*, *51*(3), 230–241.

Wang, A., Xu, H., & Ding, X. (2020). Simultaneous optimization of drug combination dose-ratio sequence with innovative design and active learning. *Advanced Therapeutics*, *3*(4), 1900135.

Wang, H., Xiao, Q., & Mandal, A. (2021). *Lhd: Latin hypercube designs (lhds)* [R package version 1.3.3]. https://CRAN.R-project.org/package=LHD

Wang, H., Xiao, Q., & Mandal, A. (2022). *La: Lioness algorithm (la)* [R package version 2.1]. https://CRAN.R-project.org/package=LA

Wang, L., Xiao, Q., & Xu, H. (2018). Optimal maximin L_1-distance latin hypercube designs based on good lattice point designs. *The Annals of Statistics*, *46*(6B), 3741–3766.

Washington, P. M., Forcelli, P. A., Wilkins, T., Zapple, D. N., Parsadanian, M., & Burns, M. P. (2012). The effect of injury severity on behavior: A phenotypic study of cognitive and emotional deficits after mild, moderate, and severe controlled cortical impact injury in mice. *Journal of neurotrauma*, *29*(13), 2283–2296.

Williams, E. (1949). Experimental designs balanced for the estimation of residual effects of treatments. *Australian Journal of Chemistry*, *2*(2), 149–168.

Williams, G., Galna, B., Morris, M. E., & Olver, J. (2010). Spatiotemporal deficits and kinematic classification of gait following a traumatic brain injury: A systematic review. *The Journal of head trauma rehabilitation*, *25*(5), 366–374.

Williams, G., Morris, M. E., Schache, A., & McCrory, P. R. (2009). Incidence of gait abnormalities after traumatic brain injury. *Archives of physical medicine and rehabilitation*, *90*(4), 587–593.

Wong, W. K., Chen, R.-B., Huang, C.-C., & Wang, W. (2015). A modified particle swarm optimization technique for finding optimal designs for mixture models. *PLoS One*, *10*(6), e0124720.

Wu, C. J., & Hamada, M. S. (2021). *Experiments: Planning, analysis, and optimization* (Vol. 736). John Wiley & Sons.

Xiao, Q., & Xu, H. (2017). Construction of maximin distance Latin squares and related Latin hypercube designs. *Biometrika*, *104*(2), 455–464.

Xiao, Q., & Xu, H. (2018). Construction of maximin distance designs via level permutation and expansion. *Statistica Sinica*, *28*(3), 1395–1414.

Xiao, Q., & Xu, H. (2021). A mapping-based universal kriging model for order-of-addition experiments in drug combination studies. *Computational Statistics & Data Analysis*, *157*, 107155.

Yang, J., & Liu, M.-Q. (2012). Construction of orthogonal and nearly orthogonal Latin hypercube designs from orthogonal designs. *Statistica Sinica*, 433–442.

Yang, X.-S. (2009). Firefly algorithms for multimodal optimization. *International symposium on stochastic algorithms*, 169–178.

Yang, X.-S. (2010). A new metaheuristic bat-inspired algorithm. *Nature inspired cooperative strategies for optimization (nicso 2010)* (pp. 65–74). Springer.

Yang, X.-S. (2020). *Nature-inspired optimization algorithms*. Academic Press.

Yang, X.-S., & Deb, S. (2009). Cuckoo search via lévy flights. *2009 World congress on nature & biologically inspired computing (NaBIC)*, 210–214.

Yao, X., Liu, Y., & Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary computation*, *3*(2), 82–102.

Ye, K. Q. (1998). Orthogonal column Latin hypercubes and their application in computer experiments. *Journal of the American Statistical Association*, *93*(444), 1430–1439.

Ye, K. Q., Li, W., & Sudjianto, A. (2000). Algorithmic construction of optimal symmetric Latin hypercube designs. *Journal of statistical planning and inference*, *90*(1), 145–159.

Zhou, Y., & Xu, H. (2015). Space-filling properties of good lattice point sets. *Biometrika*, *102*(4), 959–966.

Zhu, L. L., Lindenberg, R., Alexander, M. P., & Schlaug, G. (2010). Lesion load of the corticospinal tract predicts motor impairment in chronic stroke. *Stroke*, *41*(5), 910–915.

Ziebell, J. M., & Morganti-Kossmann, M. C. (2010). Involvement of pro-and anti-inflammatory cytokines and chemokines in the pathophysiology of traumatic brain injury. *Neurotherapeutics*, *7*(1), 22–30.