LHD: An All-encompassing R Package for Constructing Optimal Latin Hypercube Designs

by Hongzhi Wang, Qian Xiao and Abhyuday Mandal

Abstract Optimal Latin hypercube designs (LHDs), including maximin distance LHDs, maximum projection LHDs and orthogonal LHDs, are widely used in computer experiments. It is challenging to construct such designs with flexible sizes, especially for large ones, for two main reasons. One reason is that theoretical results, such as algebraic constructions ensuring the maximin distance property or orthogonality, are only available for certain design sizes. For design sizes where theoretical results are unavailable, search algorithms can generate designs. However, their numerical performance is not guaranteed to be optimal. Another reason is that when design sizes increase, the number of permutations grows exponentially. Constructing optimal LHDs is a discrete optimization process, and enumeration is nearly impossible for large or moderate design sizes. Various search algorithms and algebraic constructions have been proposed to identify optimal LHDs, each having its own pros and cons. We develop the R package `LHD` [1] which implements various search algorithms and algebraic constructions. We embedded different optimality criteria into each of the search algorithms, and they are capable of constructing different types of optimal LHDs even though they were originally invented to construct maximin distance LHDs only. Another input argument that controls maximum CPU time is added to each of the search algorithms to let users flexibly allocate their computational resources. We demonstrate functionalities of the package by using various examples, and we provide guidance for experimenters on finding suitable optimal designs. The `LHD` package is easy to use for practitioners and possibly serves as a benchmark for future developments in LHD.

1 Introduction

Computer experiments are widely used in scientific research and industrial production, where complex computer codes, commonly high-fidelity simulators, generate data instead of real physical systems (Sacks et al., 1989; Fang et al., 2005). The outputs from computer experiments are deterministic (that is, free of random errors), and therefore replications are not needed (Butler, 2001; Joseph and Hung, 2008; Ba et al., 2015). 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 and 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 (Ye, 1998; Fang et al., 2005; Joseph et al., 2015; Xiao and Xu, 2018). One key reason is that theoretical results, such as algebraic constructions which guarantee the maximin distance property or orthogonality, are only established for specific design sizes. These constructions provide theoretical guarantees on the design quality but are limited in their applicability. For design sizes where such theoretical guarantees do not exist, search algorithms can generate designs. However, the performance of search-based designs depends on the algorithm employed, the search space explored, and the computational resources allocated, meaning they cannot be guaranteed to be optimal. Constructing optimal LHDs is a discrete optimization process, where enumerating all possible solutions guarantees the optimal design for a given size. However, this approach becomes computationally infeasible as the number of permutations grows exponentially with increasing design sizes, making it another key reason that adds to the challenge.

[1]Package link on CRAN: https://cran.r-project.org/web/packages/LHD/index.html

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 paper, n denotes the run size and k denotes the factor size. A space-filling LHD has its sampled region as scattered 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 and Mitchell, 1995), the discrepancy criteria (Hickernell, 1998; Fang et al., 2002, 2005) and the entropy criterion (Shewry and Wynn, 1987). Since there are as many as $(n!)^k$ candidate LHDs for a given design size, it is nearly impossible to find the space-filling one by enumeration when n and k are moderate or large. In the current literature, both the search algorithms (Morris and Mitchell, 1995; Leary et al., 2003; Joseph and Hung, 2008; Ba et al., 2015; Ye et al., 2000; Jin et al., 2005; Liefvendahl and Stocki, 2006; Grosso et al., 2009; Chen et al., 2013) and algebraic constructions (Zhou and Xu, 2015; Xiao and Xu, 2017; Wang et al., 2018) are used to construct space-filling LHDs.

Space-filling designs often focus on the full-dimensional space. 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 a simulated annealing (SA) based algorithm. In the `LHD` package, we incorporated the MaxPro criterion with other different algorithms such as the particle swarm optimization (PSO) and genetic algorithm (GA) framework, leading to many better MaxPro LHDs; see Section 3 for examples.

Unlike space-filling LHDs that minimize the similarities among rows, orthogonal LHDs (OLHDs) are another popular type of optimal design which consider similarities among columns. For example, OLHDs have zero column-wise correlations. Algebraic constructions are available for certain design sizes (Ye, 1998; Cioppa and Lucas, 2007; Steinberg and Lin, 2006; Sun et al., 2010, 2009; Yang and Liu, 2012; Georgiou and Efthimiou, 2014; Butler, 2001; Tang, 1993; Lin et al., 2009), but there are many design sizes where theoretical results are not available. In the `LHD` package, we implemented the average absolute correlation criterion and the maximum absolute correlation criterion (Georgiou, 2009) with SA, PSO, and GA to identify both OLHDs and nearly orthogonal LHDs (NOLHDs) for almost all design sizes.

This paper introduces the R package `LHD` available on the Comprehensive R Archive Network (`https://cran.r-project.org/web/packages/LHD/index.html`), which implements some currently popular search algorithms and algebraic constructions for constructing maximin distance LHDs, Maxpro LHDs, OLHDs and NOLHDs. We embedded different optimality criteria including the maximin distance criterion, the MaxPro criterion, the average absolute correlation criterion, and the maximum absolute correlation criterion in each of the search algorithms which were originally invented to construct maximin distance LHDs only (Morris and Mitchell, 1995; Leary et al., 2003; Joseph and Hung, 2008; Liefvendahl and Stocki, 2006; Chen et al., 2013), and each of them is capable of constructing different types of optimal LHDs through the package. To let users flexibly allocate their computational resources, we also embedded an input argument that limits the maximum CPU time for each of the algorithms, where users can easily define how and when they want the algorithms to stop. An algorithm can stop in one of two ways: either when the user-defined maximum number of iterations is reached or when the user-defined maximum CPU time is exceeded. For example, users can either allow the algorithm to run for a specified number of iterations without restricting the maximum CPU time or set a maximum CPU time limit to stop the algorithm regardless of the number of iterations completed. After an algorithm is completed or stopped, the number of iterations completed along with the average CPU time per iteration will be presented to users for their information. The R package `LHD` is an integrated tool for users with little or no background in design theory, and they can easily find optimal LHDs with desired sizes. Many new designs that are better than the existing ones are discovered; see Section 3.

The remainder of the paper is organized as follows. Section 2 illustrates different optimality criteria for LHDs. Section 3 demonstrates some popular search algorithms and their implementation details in the LHD package along with examples. Section 4 discusses some useful algebraic constructions as well as examples of how to implement them via the developed package. Section 5 concludes with a summary.

2 Optimality Criteria for LHDs

Various criteria are proposed to measure designs' space-filling properties (Johnson et al., 1990; Hickernell, 1998; Fang et al., 2002). In this paper, we focus on the currently popular maximin distance criterion (Johnson et al., 1990), which seeks to scatter design points over experimental domains so that the minimum distances between points are maximized. Let **X** denote an LHD matrix throughout this paper. Define the L_q-distance between two runs x_i and x_j of **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 the design **X** as $d_q(\mathbf{X}) = \min\{d_q(x_i, x_j), 1 \le i < j \le n\}$. In this paper, we consider $q = 1$ and $q = 2$, i.e. the Manhattan (L_1) and Euclidean (L_2) distances. A design **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 design has 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:

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

where p is a tuning parameter. This ϕ_p criterion in Equation (1) is asymptotically equivalent to the Maximin distance criterion as $p \to \infty$. In practice, $p = 15$ often suffices (Morris and Mitchell, 1995). In the LHD package, the 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 subspaces 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 **X** is called a maximum projection LHD (MaxPro LHD) if it minimizes the maximum projection criterion such that

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

From Equation (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 subspaces. Note that this criterion was originally defined using design points scaled to the unit hypercube $[0,1]^k$ in Joseph et al. (2015), whereas our design points are represented as integer levels. A simple transformation can be applied to revert the scaling. For example, the transformation $\mathbf{X}_{Scaled} * n - 0.5$, can be applied, meaning that each element of every design point in scaled unit hypercube is multiplied by its run size n and then adding 0.5. The illustrative example at the end of Section 3 applies this transformation to ensure a fair comparison of performance. In the LHD package, the function MaxProCriterion() implements this criterion.

Orthogonal and nearly orthogonal designs that aim to minimize the correlations between factors are widely used in experiments (Georgiou, 2009; Steinberg and Lin, 2006; Sun and Tang, 2017). Two major correlation-based criteria to measure designs' orthogonality are the average absolute correlation criterion and the maximum absolute correlation criterion

(Georgiou, 2009), denoted as ave($|q|$) and max$|q|$, respectively:

$$\text{ave}(|q|) = \frac{2\sum_{i=1}^{k-1}\sum_{j=i+1}^{k}|q_{ij}|}{k(k-1)} \text{ and } \max|q| = \max_{i,j}|q_{ij}|, \tag{3}$$

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 a smaller ave($|q|$) or max$|q|$ are generally preferred in practice. In the LHD package, functions AvgAbsCor() and MaxAbsCor() implement the criteria ave($|q|$) and max$|q|$, respectively.

2.1 Illustrating Examples for the Introduced Optimality Criteria

This subsection demonstrates some examples of how to use the optimality criteria introduced above from the developed LHD package. To generate a random LHD matrix, the function rLHD can be used. For example,

```
> 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
```

The input arguments for the function rLHD are the run-size n and the factor size k. Continuing with the above randomly generated LHD X, we evaluate it with respect to different optimality criteria. For example,

```
> phi_p(X)              #The maximin L1-distance criterion.
[1] 0.3336608
> 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)          #The average absolute correlation criterion.
[1] 0.5333333
> MaxAbsCor(X)          #The maximum absolute correlation criterion.
[1] 0.9
```

The input arguments of the function phi_p are an LHD matrix X, p and q, where p and q come directly from the equation 1. Note that the default settings within function phi_p are $p = 15$ and $q = 1$ (the Manhattan distance) and user can change the settings. For functions MaxProCriterion, AvgAbsCor, and MaxAbsCor, there is only one input argument, which is an LHD matrix X.

3 Search Algorithms for Optimal LHDs with Flexible Sizes

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 the annealing process in metallurgy. Morris and Mitchell (1995) proposed a modified SA that randomly exchanges the elements in LHD to seek potential improvements. If such an exchange leads to a better LHD under a given optimality criterion, the exchange is maintained. Otherwise, it is kept 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. In this article, we focus on minimizing the optimality criteria outlined in Section 2, meaning only minimization optimization problems are considered. Such probability guarantees that the exchange that leads to a slightly worse LHD has a higher chance of being kept than the exchange that leads to a significantly worse LHD, because an exchange which leads to a slightly worse LHD has a lower value of $\Phi(\mathbf{X}_{new}) - \Phi(\mathbf{X})$. Such an exchange procedure will be implemented iteratively to improve LHD. When there are no improvements after certain attempts, the current temperature T is annealed. Note that a large value of $\Phi(\mathbf{X}_{new}) - \Phi(\mathbf{X})$ (exchange leading to a significantly worse LHD) is more likely to remain during the early phase of the search process when T is relatively high, and it is less likely to stay later when T decreases (annealed). The best LHD is identified after the algorithm converges or the budget constraint is reached. In the `LHD` package, the function `SA()` implements this algorithm:

```
SA(n, k, N = 10, T0 = 10, rate = 0.1, Tmin = 1, Imax = 5, OC = "phi_p",
p = 15, q = 1, maxtime = 5)
```

Table 1 provides an overview of all the input arguments in `SA()`. `n` and `k` are the desired run size and factor size. `T0` is an initial temperature, `rate` is the temperature decreasing rate, and `Tmin` is the minimum temperature. If the current temperature is smaller than `Tmin`, the current loop in the algorithm will stop and the current number of iterations will increase by one. There are two stopping criteria for the entire function: when the current number of iterations reaches the maximum (denoted as `N` in the function) or when the cumulative CPU time reaches the maximum (denoted as `maxtime` in the function), respectively. Either of those will trigger the stop of the function, whichever is earlier. For input argument `OC` (optimality criterion), “phi_p" returns maximin distance LHDs, “MaxProCriterion" returns MaxPro LHDs, and “AvgAbsCor" or “MaxAbsCor" returns orthogonal LHDs.

Table 1: Overview of Input Arguments of the `SA` Function

Argument	Description
`n`	A positive integer that defines the number of rows (or run size) of output LHD.
`k`	A positive integer that defines the number of columns (or factor size) of output LHD.
`N`	A positive integer that defines the maximum number of iterations in the algorithm. A large value of `N` will result in a high CPU time, and it is recommended to be no greater than 500. The default is set to be 10.
`T0`	A positive number that defines the initial temperature. The default is set to be 10, which means the temperature anneals from 10 in the algorithm.
`rate`	A positive percentage that defines the temperature decrease rate, and it should be in (0,1). For example, rate=0.25 means the temperature decreases by 25% each time. The default is set to be 10%.
`Tmin`	A positive number that defines the minimum temperature allowed. When current temperature becomes smaller or equal to `Tmin`, the stopping criterion for current loop is met. The default is set to be 1.
`Imax`	A positive integer that defines the maximum perturbations the algorithm will try without improvements before temperature is reduced. The default is set to be 5. For CPU time consideration, `Imax` is recommended to be no greater than 5.
`OC`	An optimality criterion. The default setting is “phi_p", and it could be one of the following: “phi_p", “AvgAbsCor", “MaxAbsCor", “MaxProCriterion".
`p`	A positive integer, which is one parameter in the ϕ_p formula, and `p` is preferred to be large. The default is set to be 15.
`q`	A positive integer, which is one parameter in the ϕ_p formula, and `q` could be either 1 or 2. If `q` is 1, the Manhattan (rectangular) distance will be calculated. If `q` is 2, the Euclidean distance will be calculated.
`maxtime`	A positive number, which indicates the expected maximum CPU time, and it is measured by minutes. For example, `maxtime`=3.5 indicates the CPU time will be no greater than three and a half minutes. The default is set to be 5.

Leary et al. (2003) modified the SA algorithm in Morris and Mitchell (1995) to search for 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 rather than a random LHD. 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, the function OASA() implements the modified SA algorithm.:

```
OASA(OA, N = 10, T0 = 10, rate = 0.1, Tmin = 1, Imax = 5, OC = "phi_p",
p = 15, q = 1, maxtime = 5),
```

where all the input arguments are the same as in SA except that OA must be an orthogonal array.

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 that has the largest average pairwise correlations with all other columns. Next, the algorithm will select the row which has the largest total row-wise distance with all other rows. Then, the element at the selected row and 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). In the LHD package, the function SA2008() implements this algorithm:

```
SA2008(n, k, N = 10, T0 = 10, rate = 0.1, Tmin = 1, Imax = 5, OC = "phi_p",
p = 15, q = 1, maxtime = 5),
```

where all the input arguments are the same as in SA.

3.2 Particle Swarm Optimization Algorithms

Particle swarm optimization (PSO, Kennedy and Eberhart (1995)) is a metaheuristic optimization algorithm inspired by the social behaviors of animals. Recent research (Chen et al., 2013) adapted the classic PSO algorithm and proposed LaPSO to identify maximin distance LHDs. Since this is a discrete optimization task, LaPSO redefines the steps in which each particle updates its velocity and position in the general PSO framework. In the LHD package, the function LaPSO() implements this algorithm:

```
LaPSO(n, k, m = 10, N = 10, SameNumP = 0, SameNumG = n/4, p0 = 1/(k - 1),
OC = "phi_p", p = 15, q = 1, maxtime = 5)
```

Table 2 provides an overview of all the input arguments in LaPSO(), where n, k, N, OC, p, q, and maxtime are exactly the same as the input arguments in the function SA(). m is the number of particles, which represents candidate solutions in the PSO framework. SameNumP and SameNumG are two tuning parameters that denote how many exchanges would be performed to reduce the Hamming distance towards the personal best and the global best. p0 is the tuning parameter that denotes the probability of a random swap for two elements in the current column of the current particle to prevent the algorithm from being stuck at the local optimum. In 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 $2/(k-1)$. The stopping criterion of the function is the same as that of the function SA.

3.3 Genetic Algorithms

The genetic algorithm (GA) is a nature-inspired metaheuristic optimization algorithm that mimics Charles Darwin's idea of natural selection (Holland et al., 1992; Goldberg, 1989). Liefvendahl and Stocki (2006) proposed a version of GA for identifying maximin distance LHDs. They implement the column exchange technique to solve the discrete optimization task. In the LHD package, the function GA() implements this algorithm:

Table 2: Overview of Input Arguments of the `LaPSO` Function

Argument	Description
`n`	A positive integer that defines the number of rows (or run size) of output LHD.
`k`	A positive integer that defines the number of columns (or factor size) of output LHD.
`m`	A positive integer that defines the number of particles, where each particle is a candidate solution. A large value of `N` will result in a high CPU time, and it is recommended to be no greater than 100. The default is set to be 10.
`N`	A positive integer that defines the maximum number of iterations in the algorithm. A large value of `N` will result in a high CPU time, and it is recommended to be no greater than 500. The default is set to be 10.
`SameNumP`	A non-negative integer that defines how many elements in current column of current particle should be the same as corresponding Personal Best. SameNumP can be 0, 1, 2, . . . , n, where 0 means to skip the element exchange, which is the default setting.
`SameNumG`	A non-negative integer that defines how many elements in current column of current particle should be the same as corresponding Global Best. SameNumG can be 0, 1, 2, . . . , n, where 0 means to skip the element exchange. The default setting is n/4. Note that SameNumP and SameNumG cannot be 0 at the same time.
`p0`	A probability of exchanging two randomly selected elements in current column of current particle LHD. The default is set to be 1/(k - 1).
`OC`	An optimality criterion. The default setting is "phi_p", and it could be one of the following: "phi_p", "AvgAbsCor", "MaxAbsCor", "MaxProCriterion".
`p`	A positive integer, which is one parameter in the ϕ_p formula, and `p` is preferred to be large. The default is set to be 15.
`q`	A positive integer, which is one parameter in the ϕ_p formula, and `q` could be either 1 or 2. If `q` is 1, the Manhattan (rectangular) distance will be calculated. If `q` is 2, the Euclidean distance will be calculated.
`maxtime`	A positive number, which indicates the expected maximum CPU time, and it is measured by minutes. For example, `maxtime`=3.5 indicates the CPU time will be no greater than three and a half minutes. The default is set to be 5.

```
GA(n, k, m = 10, N = 10, pmut = 1/(k - 1), OC = "phi_p", p = 15, q = 1,
maxtime = 5)
```

Table 3 provides an overview of all the input arguments in `GA()`, where `n`, `k`, `N`, `OC`, `p`, `q`, and `maxtime` are exactly the same as the input arguments in the function `SA()`. `m` is the population size, which represents how many candidate solutions in each iteration, and must be an even number. `pmut` is the tuning parameter that controls how likely the mutation would happen. When mutation occurs, two randomly selected elements will be exchanged in the current column of the current LHD. `pmut` serves the same purpose as `p0` in `LaPSO()`, which prevents the algorithm from getting stuck at the local optimum, and it is recommended to be $1/(k-1)$. The stopping criterion of the function is the same as that of the function `SA`.

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. In Table 4, we summarize the R functions of the 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 the 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
```

Table 3: Overview of Input Arguments of the GA Function

Argument	Description
n	A positive integer that defines the number of rows (or run size) of output LHD.
k	A positive integer that defines the number of columns (or factor size) of output LHD.
m	A positive even integer, which stands for the population size and it must be an even number. The default is set to be 10. A large value of m will result in a high CPU time, and it is recommended to be no greater than 100.
N	A positive integer that defines the maximum number of iterations in the algorithm. A large value of N will result in a high CPU time, and it is recommended to be no greater than 500. The default is set to be 10.
pmut	A probability for mutation. When the mutation happens, two randomly selected elements in current column of current LHD will be exchanged. The default is set to be 1/(k - 1).
OC	An optimality criterion. The default setting is "phi_p", and it could be one of the following: "phi_p", "AvgAbsCor", "MaxAbsCor", "MaxProCriterion".
p	A positive integer, which is one parameter in the ϕ_p formula, and p is preferred to be large. The default is set to be 15.
q	A positive integer, which is one parameter in the ϕ_p formula, and q could be either 1 or 2. If q is 1, the Manhattan (rectangular) distance will be calculated. If q is 2, the Euclidean distance will be calculated.
maxtime	A positive number, which indicates the expected maximum CPU time, and it is measured by minutes. For example, maxtime=3.5 indicates the CPU time will be no greater than three and a half minutes. The default is set to be 5.

Table 4: Search algorithm functions in the LHD package

Function	Description
SA	Returns an LHD via the simulated annealing algorithm (Morris and 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 and Hung, 2008).
LaPSO	Returns an LHD via the particle swarm optimization (Chen et al., 2013).
GA	Returns an LHD via the genetic algorithm (Liefvendahl and Stocki, 2006).

```
     [,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

#Similarly, generations of 5 by 3 maximin distance LHD by the SA2008, LaPSO and GA functions.
> try.SA2008 = SA2008(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), an orthogonal array with 9 runs, 2 factors, 3 levels, and 2 strength.
> 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" (that is, the maximin distance criterion), leading 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 of different argument settings.

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

#Below try.SA2008 is a 5 by 3 maximin distance LHD generated by SA with
#the multi-objective optimization approach. The input arguments are interpreted
#the same as the design try.SA above.
> 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)

#Below try.OASA 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 input 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)

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

#Below try.GA is a 5 by 3 OLHD generated by the genetic algorithm with the
#population size 20 (m = 20), number of iterations 30 (N = 30),  mutation
#probability 0.5 (pmut = 0.5), maximum absolute correlation criterion
#(OC = "MaxAbsCor"), and maximum CPU time 5 minutes (maxtime = 5).
> try.GA = GA(n = 5, k = 3, m = 20, N = 30, pmut = 0.5, OC = "MaxAbsCor",
+                 maxtime = 5); try.GA
     [,1] [,2] [,3]
[1,]    2    1    2
[2,]    4    4    5
[3,]    3    5    1
[4,]    5    2    3
[5,]    1    3    4
#Recall the value is 0.9 from the random LHD in Section 2.
> MaxAbsCor(try.GA)
[1] 0.1     #The maximum absolute correlation between columns is 0.1
```

Next, we discuss some details of the implementation. In SA based algorithms (`SA`, `SA2008`, and `OASA`), the number of iterations `N` is recommended to be no greater than 500 for computing time considerations. The input `rate` determines the percentage of the decrease in current temperature (for example, 0.1 means a decrease of 10% each time). A high rate would make the temperature rapidly drop, 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 attempt 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 the `"phi_p"` criterion in (1) is set as default. `OC` has other options, including `"MaxProCriterion"`, `"AvgAbsCor"` and `"MaxAbsCor"`. Our algorithms support both the L_1 and L_2 distances.

For every algorithm, we incorporate a progress bar to visualize the computing time used. After an algorithm is completed, information on "average CPU time per iteration" and "numbers of iterations completed" will be presented. Users can set the limit for the CPU time used for each algorithm using the argument `maxtime`, according to their practical needs.

We also provide some illustrative code to demonstrate that the designs found in the `LHD` package are better than the existing ones, and the code in the following can be easily modified to construct other design sizes or other LHD types. Out of 100 trials, the code below shows the GA in the `LHD` package constructed better MaxPro LHDs 99 times compared to the algorithm in the `MaxPro` package, when 500 iterations are set for both algorithms. We did not compare the CPU time between these two packages since one is written in the R environment and the other one is written in the C++ environment, but with the same number of iterations, the GA in the `LHD` package almost always constructs better MaxPro LHDs.

```
#Make sure both packages are properly installed before load them
> library(LHD)
> library(MaxPro)

> count = 0 #Define a variable for counting purpose

> k = 5       #Factor size 5
> n = 10*k    #Run size = 10*factor size
```

```
#Setting 500 iterations for both algorithms, below loop counts
#how many times the GA from LHD package outperforms the algorithm
#from MaxPro package out of 100 times
> for (i in 1:100) {

  LHD = LHD::GA(n = n, k = k, m = 100, N = 500)
  MaxPro = MaxPro::MaxProLHD(n = n, p = k, total_iter = 500)$Design

  #MaxPro * n + 0.5 applied the transformation mentioned in Section 2
  #to revert the scaling.
  Result.LHD = LHD::MaxProCriterion(LHD)
  Result.MaxPro = LHD::MaxProCriterion(MaxPro * n + 0.5)

  if (Result.LHD < Result.MaxPro) {count = count + 1}

}

> count
[1] 99
```

4 Algebraic Constructions for Optimal LHDs with Certain Sizes

There are algebraic constructions available for certain design sizes, and theoretical results are developed to guarantee the efficiency of such designs. Algebraic constructions almost do not require any searching, which are especially attractive for large designs. In this section, we present algebraic constructions that are available in the LHD package for maximin distance LHDs and orthogonal LHDs.

4.1 Algebraic Constructions for Maximin Distance LHDs

Wang et al. (2018) proposed to generate maximin distance LHDs via good lattice point (GLP) sets (Zhou and Xu, 2015) and Williams transformation (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. This construction method by Wang et al. (2018) is very attractive for constructing large maximin distance LHDs. In the LHD package, function FastMmLHD() implements this method:

```
FastMmLHD(n, k, method = "manhattan", t1 = 10),
```

where n and k are the desired run size and factor size. method is a distance measure method which can be one of the following: “euclidean", “maximum", “manhattan", “canberra", “binary" or “minkowski". Any unambiguous substring can be given. t1 is a tuning parameter, which determines how many repeats will be implemented to search for the optimal design. The default is set to be 10.

Tang (1993) proposed to construct orthogonal array-based LHDs (OALHDs) from existing orthogonal arrays (OAs), and 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:

```
OA2LHD(OA),
```

where OA is an orthogonal array matrix. Users only need to input an OA and the function will return an OALHD with the same design size as the input OA.

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 a rich literature on the constructions of OLHDs with various design sizes, but they are often too hard for practitioners to replicate in practice. The LHD package implements some currently popular methods (Ye, 1998; Cioppa and Lucas, 2007; Sun et al., 2010; Tang, 1993; Lin et al., 2009; Butler, 2001) for practitioners and the functions are easy to use.

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. In the LHD package, function `OLHD.Y1998()` implements this algebraic construction:

```
OLHD.Y1998(m),
```

where input argument `m` is the m in the construction of Ye (1998). 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. In the LHD package, function `OLHD.C2007()` implements this algebraic construction with input argument `m` remaining the same:

```
OLHD.C2007(m)
```

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. In the LHD package, function `OLHD.S2010()` implements this algebraic construction:

```
OLHD.S2010(C, r, type = "odd"),
```

where input arguments `C` and `r` are c and r in the construction. When input argument `type` is `"odd"`, the output design size would be $n = r2^{c+1} + 1$ by $k = 2^c$. When input argument `type` is `"even"`, the output design size would be $n = r2^{c+1}$ by $k = 2^c$.

Lin et al. (2009) constructed OLHDs or NOLHDs with n^2 runs and $2fp$ factors by coupling OLHD(n, p) or NOLHD(n, p) with an OA($n^2, 2f, n, 2$). For example, an OLHD(11, 7), coupled with an OA(121,12,11,2), would yield an OLHD(121, 84). The design size of output OLHD or NOLHD highly depends on the existence of the OAs. In the LHD package, function `OLHD.L2009()` implements this algebraic construction:

```
OLHD.L2009(OLHD, OA),
```

where input arguments `OLHD` and `OA` are the OLHD and OA to be coupled, and their design sizes need to be aligned with the designated pattern of the construction.

Butler (2001) proposed a method to construct OLHDs with the run size n being odd primes and factor size k being less than or equal to $n - 1$ via the Williams transformation (Williams, 1949). In the LHD package, function `OLHD.B2001()` implements this algebraic construction with input arguments `n` and `k` exactly matching those in construction:

```
OLHD.B2001(n, k)
```

4.3 Illustrating Examples for the Implemented Algebraic Constructions

In Table 5, we summarize the algebraic constructions 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.

Table 5: Algebraic constructions in the LHD package

Function	Description
FastMmLHD	Returns a maximin distance LHD matrix (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 and 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 (Lin et al., 2009).
OLHD.B2001	Returns an orthogonal LHD (Butler, 2001) with the run size n being odd primes and factor size k being less than or equal to $n - 1$.

```
#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
[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,
1,2,3,4,5,2,1,3,5,2,4,3,1,1,1,1,1,1,4,3,2,1,5,5,5,5,5,5,1,4,4,4,4,4,1,
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

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

5 Other R Packages for Latin Hypercube and Comparative Discussion

Several R packages have been developed to facilitate Latin hypercube samples and design constructions for computer experiments. Among these, the `lhs` package (Carnell, 2024) is widely recognized for its utility. It provides functions for generating both random and optimized Latin hypercube samples (but not designs), and its methods are particularly useful for simulation studies where space-filling properties are desired but design optimality is not the primary focus. The `SLHD` package (Ba, 2015) was originally developed for generating sliced LHDs (Ba et al., 2015), while practitioners can set the number of slices to one to use the package for generating maximin LHDs. The `MaxPro` package (Ba and Joseph, 2018) focuses on constructing designs that maximize projection properties. One of its functions, `MaxProLHD`, generates MaxPro LHDs using a simulated annealing algorithm (Joseph et al., 2015).

While we acknowledge the contributions of other relevant R packages, we emphasize the distinguishing features of our developed package. The `LHD` package embeds multiple optimality criteria, enabling the construction of various types of optimal LHDs. In contrast, `lhs` and `SLHD` primarily focus on space-filling Latin hypercube samples and designs, while `MaxPro` primarily focuses on maximum projection LHDs. The `LHD` package implements various search algorithms and algebraic constructions, whereas the other three packages do not implement algebraic constructions, and both `SLHD` and `MaxPro` only implement one algorithm to construct LHDs. The primary application of `LHD` is in the design of computer experiments, whereas `lhs` is mainly used for sampling and simulation studies. Therefore, `LHD` emphasizes design optimality, while `lhs` emphasizes the space-filling properties of samples.

6 Conclusion and Recommendation

`LHD` package implements popular search algorithms, including the SA (Morris and Mitchell, 1995), OASA (Leary et al., 2003), SA2008 (Joseph and Hung, 2008), LaPSO (Chen et al., 2013) and GA (Liefvendahl and Stocki, 2006), along with some widely used algebraic constructions (Wang et al., 2018; Ye, 1998; Cioppa and Lucas, 2007; Sun et al., 2010; Tang, 1993; Lin et al., 2009; Butler, 2001), for constructing three types of commonly used optimal LHDs: the maximin distance LHDs, the maximum projection LHDs and the (nearly) orthogonal LHDs. 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.

Among very few R libraries particularly for LHDs, `LHD` is comprehensive and self-contained as it not only has search algorithms and algebraic constructions, but also has other useful functions for LHD research and development such as calculating different optimality criteria, generating random LHDs, exchanging two random elements in a matrix, and calculating intersite distance between matrix rows. The help manual in the package documentation contains further details and illustrative examples for users who want to explore more of the functions in the package.

Acknowledgments

This research was partially supported by the National Science Foundation (NSF) grant DMS-2311186 and the National Key R&D Program of China 2024YFA1016200. The authors appreciate the reviewers' constructive comments and suggestions.

References

S. Ba. *SLHD: Maximin-Distance (Sliced) Latin Hypercube Designs*, 2015. URL `https://CRAN.R-project.org/package=SLHD`. R package version 2.1-1. [p14]

S. Ba and V. R. Joseph. *MaxPro: Maximum Projection Designs*, 2018. URL `https://CRAN.R-project.org/package=MaxPro`. R package version 4.1-2. [p14]

S. Ba, W. R. Myers, and W. A. Brenneman. Optimal sliced Latin hypercube designs. *Technometrics*, 57(4):479–487, 2015. [p1, 2, 14]

N. A. Butler. Optimal and orthogonal Latin hypercube designs for computer experiments. *Biometrika*, 88(3):847–857, 2001. [p1, 2, 12, 13, 15]

R. Carnell. *lhs: Latin Hypercube Samples*, 2024. URL `https://CRAN.R-project.org/package=lhs`. R package version 1.2.0. [p14]

R.-B. Chen, D.-N. Hsieh, Y. Hung, and W. Wang. Optimizing Latin hypercube designs by particle swarm. *Statistics and computing*, 23(5):663–676, 2013. [p2, 6, 8, 15]

T. M. Cioppa and T. W. Lucas. Efficient nearly orthogonal and space-filling Latin hypercubes. *Technometrics*, 49(1):45–55, 2007. [p2, 12, 13, 15]

K.-T. Fang, C.-X. Ma, and P. Winker. Centered L_2-discrepancy of random sampling and Latin hypercube design, and construction of uniform designs. *Mathematics of Computation*, 71(237):275–296, 2002. [p2, 3]

K.-T. Fang, R. Li, and A. Sudjianto. *Design and modeling for computer experiments*. CRC press, 2005. [p1, 2]

S. D. Georgiou. Orthogonal Latin hypercube designs from generalized orthogonal designs. *Journal of Statistical Planning and Inference*, 139(4):1530–1540, 2009. [p2, 3, 4]

S. D. Georgiou and I. Efthimiou. Some classes of orthogonal Latin hypercube designs. *Statistica Sinica*, 24(1):101–120, 2014. [p2]

D. E. Goldberg. Genetic algorithms in search. *Optimization, and MachineLearning*, 1989. [p6]

A. Grosso, A. Jamali, and M. Locatelli. Finding maximin Latin hypercube designs by iterated local search heuristics. *European Journal of Operational Research*, 197(2):541–547, 2009. [p2]

F. Hickernell. A generalized discrepancy and quadrature error bound. *Mathematics of computation*, 67(221):299–322, 1998. [p2, 3]

J. H. Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992. [p6]

R. Jin, W. Chen, and A. Sudjianto. An efficient algorithm for constructing optimal design of computer experiments. *Journal of statistical planning and inference*, 134(1):268–287, 2005. [p2, 3]

M. E. Johnson, L. M. Moore, and D. Ylvisaker. Minimax and maximin distance designs. *Journal of statistical planning and inference*, 26(2):131–148, 1990. [p2, 3]

V. R. Joseph and Y. Hung. Orthogonal-maximin Latin hypercube designs. *Statistica Sinica*, pages 171–186, 2008. [p1, 2, 6, 8, 15]

V. R. Joseph, E. Gul, and S. Ba. Maximum projection designs for computer experiments. *Biometrika*, 102(2):371–380, 2015. [p1, 2, 3, 14]

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995. [p6]

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983. [p4]

S. Leary, A. Bhaskar, and A. Keane. Optimal orthogonal-array-based Latin hypercubes. *Journal of Applied Statistics*, 30(5):585–598, 2003. [p2, 5, 6, 8, 15]

M. Liefvendahl and R. Stocki. A study on algorithms for optimization of Latin hypercubes. *Journal of statistical planning and inference*, 136(9):3231–3247, 2006. [p2, 6, 8, 15]

C. D. Lin, R. Mukerjee, and B. Tang. Construction of orthogonal and nearly orthogonal Latin hypercubes. *Biometrika*, 96(1):243–247, 2009. [p2, 12, 13, 15]

M. D. McKay, R. J. Beckman, and W. J. Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21 (2):239–245, 1979. [p1]

M. D. Morris and T. J. Mitchell. Exploratory designs for computational experiments. *Journal of statistical planning and inference*, 43(3):381–402, 1995. [p2, 3, 4, 6, 8, 15]

J. Sacks, S. B. Schiller, and W. J. Welch. Designs for computer experiments. *Technometrics*, 31 (1):41–47, 1989. [p1]

M. C. Shewry and H. P. Wynn. Maximum entropy sampling. *Journal of applied statistics*, 14 (2):165–170, 1987. [p2]

D. M. Steinberg and D. K. J. Lin. A construction method for orthogonal Latin hypercube designs. *Biometrika*, 93(2):279–288, 2006. [p2, 3]

F. Sun and B. Tang. A general rotation method for orthogonal Latin hypercubes. *Biometrika*, 104(2):465–472, 2017. [p3]

F. Sun, M.-Q. Liu, and D. K. J. Lin. Construction of orthogonal Latin hypercube designs. *Biometrika*, 96(4):971–974, 2009. [p2, 12]

F. Sun, M.-Q. Liu, and D. K. J. Lin. Construction of orthogonal Latin hypercube designs with flexible run sizes. *Journal of Statistical Planning and Inference*, 140(11):3236–3242, 2010. [p2, 12, 13, 15]

B. Tang. Orthogonal array-based Latin hypercubes. *Journal of the American statistical association*, 88(424):1392–1397, 1993. [p2, 6, 11, 12, 13, 15]

L. Wang, Q. Xiao, and H. Xu. Optimal maximin L_1-distance latin hypercube designs based on good lattice point designs. *The Annals of Statistics*, 46(6B):3741–3766, 2018. [p2, 11, 13, 15]

E. Williams. Experimental designs balanced for the estimation of residual effects of treatments. *Australian Journal of Chemistry*, 2(2):149–168, 1949. [p11, 12]

Q. Xiao and H. Xu. Construction of maximin distance Latin squares and related Latin hypercube designs. *Biometrika*, 104(2):455–464, 2017. [p2]

Q. Xiao and H. Xu. Construction of maximin distance designs via level permutation and expansion. *Statistica Sinica*, 28(3):1395–1414, 2018. [p1]

J. Yang and M.-Q. Liu. Construction of orthogonal and nearly orthogonal Latin hypercube designs from orthogonal designs. *Statistica Sinica*, pages 433–442, 2012. [p2]

K. Q. Ye. Orthogonal column Latin hypercubes and their application in computer experiments. *Journal of the American Statistical Association*, 93(444):1430–1439, 1998. [p1, 2, 12, 13, 15]

K. Q. Ye, W. Li, and A. Sudjianto. Algorithmic construction of optimal symmetric Latin hypercube designs. *Journal of statistical planning and inference*, 90(1):145–159, 2000. [p2]

Y. Zhou and H. Xu. Space-filling properties of good lattice point sets. *Biometrika*, 102(4): 959–966, 2015. [p2, 11]

Hongzhi Wang
wanghongzhi.ut@gmail.com

Qian Xiao
Department of Statistics, School of Mathematical Sciences, Shanghai Jiao Tong University
800 Dongchuan Road, Minhang, Shanghai, 200240
China
qian.xiao@sjtu.edu.cn

Abhyuday Mandal
Department of Statistics, University of Georgia
310 Herty Drive, Athens, GA 30602
USA
amandal@stat.uga.edu