Statistica Sinica

An integer programming algorithm for constructing maximin distance designs from good lattice point sets

Alan R. Vazquez and Hongquan Xu

University of Arkansas and University of California, Los Angeles

Abstract: Computer experiments can build computationally-cheap statistical models to study complex computer models. These experiments are commonly conducted using maximin distance Latin hypercube designs (LHDs), which are generated using heuristic algorithms or algebraic methods in the literature. However, the performance of these algorithms deteriorates as the number of factors increases, while the algebraic methods only work for numbers of runs that are of a special kind, say, a prime number. To overcome these limitations, we introduce an integer programming algorithm to construct maximin distance LHDs of flexible sizes. Our algorithm leverages the recent advances in the field of optimization as implemented in commercial optimization solvers. Moreover, it benefits from the attractive algebraic structures given by good lattice point sets and the Williams' transformation. Using comprehensive numerical experiments, we show that, with few exceptions, our integer programming algorithm outperforms benchmark algorithms and methods for constructing LHDs with up to 113 runs.

Key words and phrases: Exact algorithm, Gaussian process, Gurobi, L_1-distance, level permutation, space-filling design.

1. Introduction

Computer experiments permit the study of complex systems that are simulated using computer models (Fang et al., 2006; Santner et al., 2018). A computer model uses algorithms and sets of mathematical equations to provide the best representation possible of the link between the input factors and responses of the system. However, many of these models are computationally expensive since they involve solving complicated partial differential equations numerically. Therefore, one of the main goals of computer experiments is to build an efficient, computationally-cheap surrogate model that approximates the computer model well. To this end, they demand cost-effective experimental designs that gather high-quality data from the computer model, using a limited number of runs.

Space-filling designs are attractive for computer experiments because their runs are conducted at points that fill the experimental region evenly. A space-filling design can be constructed by maximizing the minimum distance between its points or, alternatively, minimizing the maximum distance between its points and all other points in the region (Johnson et al., 1990). Designs that achieve the former and latter objectives are called maximin and minimax distance designs, respectively. A different construction method for space-filling designs involves minimizing a discrepancy function, which measures the distance between the empirical distribution of the design points and the uniform distribution over the entire region (Fang et al., 2000). Another construction method involves the so-called total potential energy function, the minimization of which results in design points that are as apart as possible, but in a way that they follow a user-specified distribution (Joseph et al., 2015). Pronzato and Müller (2012) provide a comprehensive review of space-filling designs generated in other ways. In this article, we adopt the maximin distance criterion and construct space-filling designs for computer experiments with many input factors.

The surrogate for a computer model commonly involves a (stationary) Gaussian process. The key component of this process is the covariance function describing the covariance between any two responses in terms of the distance between their corresponding design points. Johnson et al. (1990) show that, when these covariances rapidly decrease as the distance between the points increase, maximin distance designs are asymptotically D-optimal under a Gaussian process.

To construct maximin distance designs, it is attractive to restrict to the class of Latin hypercube designs (LHDs), since they fill the domain of each individual factor uniformly. This class also reduces the search space of maximin distance designs. There are several algorithms to construct LHDs which use metaheuristics such as simulated annealing (Morris and Mitchell, 1995; Ba et al., 2015), particle swarm optimization (Chen et al., 2013), iterated local search (Grosso et al., 2009), genetic algorithms (Liefvendahl and Stocki, 2006), evolutionary methods (Jin et al., 2005), and multi-start methods (Ye et al., 2000; Moon et al., 2011). Although these algorithms do not guarantee the optimality of the LHDs, they can generate attractive designs with up to 300 runs and up to 30 factors. However, for larger numbers of factors or runs, their performance in terms of design quality or computing time deteriorates, since these problems are challenging.

To overcome the limitations of these algorithms, several authors have introduced algebraic methods to construct large maximin distance LHDs, which use combinatorial structures such as orthogonal and nearly-orthogonal arrays (Xiao and Xu, 2018), good lattice point sets (Zhou and Xu, 2015), and Costas arrays (Xiao and Xu, 2017). Attractive complements of these methods are linear permutations (Zhou and Xu, 2015) and the Williams' transformation (Wang et al., 2018), to further improve the LHDs obtained from these structures. The algebraic construction methods are, however, only available for specific numbers of runs and factors, which prevents them from being flexible.

Computer models with a large number of factors are common in practice. For example, McKay (1995) describes a 84-factor simulator for the flow of a material in an ecosystem, and a 36-factor simulator for the environmental impact of severe accidents at nuclear power plants. Houston et al. (2001) discuss a 65-factor simulator for the management dynamics in software development. It is therefore important to develop good construction methods for maximin distance LHDs that can address these situations.

In this article, we introduce an elegant algorithm rooted in integer programming (Wolsey, 2020) to construct flexible LHDs that optimize the maximin distance criterion. Our algorithm, called IP, has two key elements. The first one is a candidate set of attractive columns from which to obtain the designs. We generate this set by concatenating the LHDs constructed by Wang et al. (2018) and then removing fully correlated columns as identified by novel theoretical results. We choose these LHDs because they have a good performance in terms of the maximin distance criterion. The second element of the IP algorithm is a problem formulation which—when supplied to state-of-the-art optimization solvers such as Gurobi, CPLEX or MOSEK—identifies the candidate columns that form the optimal LHD. The use of optimization solvers allows our algorithm to leverage the recent advances in the theory and practice of integer programming; see Bixby (2012) and Achterberg and Wunderling (2013). For a given candidate set, the solvers not only provide probably optimal LHDs but also an upper bound on the maximin distance criterion.

Using numerical experiments, we demonstrate that the IP algorithm is computationally-effective for design problems with up to 30 runs and up to 29 factors. Moreover, with few exceptions, it matches or improves upon benchmark algorithms and algebraic methods in the literature.

To tackle larger-sized design problems, we modify the IP algorithm in two ways. First, we use a smaller candidate set whose columns have a prime number of elements. Second, we implement a systematic method to remove rows from the optimal LHD obtained from this candidate set, so as to obtain LHDs with flexible run sizes. We show that these modifications allow the IP algorithm to outperform benchmark algorithms for design problems with 34 to 72 factors and 44 to 97 runs, as well as for problems with 10 and 11 factors and 101 to 113 runs. To the best of our knowledge, the IP algorithm is the first integer-programming-based approach for constructing maximin distance LHDs of practically-relevant sizes.

The rest of the article is organized as follows. Section 2 introduces background notation and concepts, and Section 3 reviews the method of Wang et al. (2018) for constructing LHDs. Section 4 presents the IP algorithm and a comprehensive comparison with benchmark methods available in the literature. Section 5 shows the modifications to the IP algorithm and their evaluation using numerical experiments. Section 6 concludes the article with remarks and directions for future research.

2. Preliminaries

We denote the integer part of x as $\lfloor x \rfloor$, the set of positive integers as $\mathbb{Z}^+$, and $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$. For a matrix $\mathbf{Y} = (y_{i,j})$ with $y_{i,j} \in \mathbb{Z}_N$, the entries of the linearly permuted matrix $\mathbf{Y} + b \pmod{N}$ are $y_{i,j} + b \pmod{N}$.

An n-factor N-run LHD $\mathbf{X} = (x_{i,j})$ is an $N \times n$ matrix in which each column is a permutation of the elements in $\mathbb{Z}_N$. We denote the i-th row and j-th column of $\mathbf{X}$ as $\mathbf{x}_i$ and $\mathbf{x}^{(j)}$, respectively.

Let $d(x_{i,u}, x_{j,u}) = |x_{i,u} - x_{j,u}|$, where $x_{i,u}$ is the i-th element of $\mathbf{x}^{(u)}$, $u = 1, \ldots, n$. For each $\mathbf{x}^{(u)}$, we define an $N(N-1)/2 \times 1$ vector of absolute element-wise distances

$$\mathbf{a}^{(u)} = \left(d(x_{1,u}, x_{2,u}), d(x_{1,u}, x_{3,u}), \ldots, d(x_{N-1,u}, x_{N,u})\right)^T.$$

We define the *distance matrix* as $\mathbf{A} = \left[\mathbf{a}^{(1)}; \mathbf{a}^{(2)}; \cdots; \mathbf{a}^{(n)}\right]$, which collects the absolute element-wise distance vectors of all columns in $\mathbf{X}$. Let $\mathbf{A}_q = (a_{i,j}^q)$ with $a_{i,j}$ denoting the entries of $\mathbf{A}$ and q a positive integer. The L_q-distances between any two distinct rows in $\mathbf{X}$ are given by the element-wise q-th root of $\mathbf{A}_q \mathbf{1}_n$, where $\mathbf{1}_n$ is an $n \times 1$ vector of ones. The distance matrix is a key component of our method to generate LHDs.

The L_q-distance of an LHD $\mathbf{X}$, denoted as $d^q(\mathbf{X})$, is the minimum L_q-distance between any two distinct rows of the design. That is,

$$d^q(\mathbf{X}) = \min\left\{ \left(\sum_{j=1}^{n} a_{i,j}^q \right)^{1/q} : i = 1, \ldots, N(N-1)/2 \right\}.$$

When comparing two LHDs, the one with the largest minimum L_q-distance between any two distinct rows is preferred according to the maximin distance criterion. An LHD that maximizes $d^q(\mathbf{X})$ is called a maximin L_q-distance LHD (Johnson et al., 1990). Here, we set $q = 1$, thereby adopting the L_1-distance. However, our methodology works for other values of q.

Two vectors are fully correlated if the correlation between them is either 1 or -1. The following lemma shows that fully correlated vectors induce the same absolute element-wise distance vectors.

Lemma 1. *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be* $N \times 1$ *vectors whose elements are permutations of* $\mathbb{Z}_N$*. Let* $\mathbf{a}_x$ *and* $\mathbf{a}_y$ *be the* $N(N-1)/2 \times 1$ *distance vectors constructed from* $\mathbf{x}$ *and* $\mathbf{y}$*, respectively. If* $\mathbf{y} = (N-1)\mathbf{1}_N - \mathbf{x}$*, then* $\mathbf{a}_x = \mathbf{a}_y$*.*

3. Construction methods based on good lattice point sets and the Williams' transformation

We now review the method of Wang et al. (2018) to construct LHDs using good lattice point sets (Zhou and Xu, 2015), linear permutations and the Williams' transformation (Williams, 1949). We also provide new theoretical results to characterize these LHDs. Supplementary Section S1 contains the proofs of these and other results in the article.

3.1 Good lattice point sets and linear permutations

Let $H = \{h_1, \ldots, h_n\}$ be a set of positive integers smaller than and coprime to N, such that $h_1 < h_2 < \cdots < h_n$. An $N \times n$ good lattice point (GLP) set $\mathbf{X}$ has elements $x_{i,j} = ih_j \pmod N$ for $i = 1, \ldots, N$, and $j = 1, \ldots, n$; see Zhou and Xu (2015). The last row of $\mathbf{X}$ is a vector of zeros. Each column of $\mathbf{X}$ is a permutation of the elements in $\mathbb{Z}_N$. Therefore, a GLP set is an LHD. We can construct an $N \times n$ GLP set for any $n \leq \phi(N)$, where $\phi(N)$ is the number of positive integers smaller than and coprime to N. We assume that $N > 3$ and so, $\phi(N)$ must be even. If N is a prime, $\phi(N) = N - 1$.

Zhou and Xu (2015) show that linear permutations of the columns of a GLP set $\mathbf{X}$ may produce a better LHD in terms of the L_1-distance. More specifically, they prove that $\mathbf{X}_b = \mathbf{X} + b \pmod N$ is an LHD with

$d^1(\mathbf{X}_b) \geq d^1(\mathbf{X})$, for $b \in \mathbb{Z}_N$. When N is a prime and $n = N - 1$, the LHDs $\mathbf{X}_b$ with the optimal value of b are competitive with those obtained by the simulated annealing (SA) algorithm of Ba et al. (2015), in terms of the L_1-distance. Moreover, the former are computationally-cheaper to generate than the latter.

3.2 The Williams' transformation and some theoretical results

Wang et al. (2018) show that the performance of the linearly permuted GLP sets can be further improved using the Williams' transformation (Williams, 1949). For an integer N and $y \in \mathbb{Z}_N$, the Williams' transformation is

$$W(y) = \begin{cases} 2y & \text{for } 0 \leq y < N/2; \\ 2(N-y)-1 & \text{for } N/2 \leq y < N. \end{cases}$$

This transformation is a permutation of elements in $\mathbb{Z}_N$. So, for an LHD $\mathbf{X}$, $W(\mathbf{X}) = (W(x_{i,j}))$ is an LHD too.

Given a GLP set $\mathbf{X}$, Wang et al. (2018) propose to evaluate all LHDs $\mathbf{Z}_b = W(\mathbf{X}_b)$ for $b = 0, \ldots, N-1$, and select the best one in terms of the L_1-distance. However, it turns out that not all the designs have to be evaluated. To see this, we first introduce two lemmas that state a relationship between the columns in the LHDs $\mathbf{Z}_b$, when N is even or odd.

Lemma 2. *Let N be even, $n = \phi(N)$, $\mathbf{X}$ be an $N \times n$ GLP set, $\mathbf{X}_b = \mathbf{X} + b \pmod N$ and $\mathbf{Z}_b = W(\mathbf{X}_b)$, with $b \in \mathbb{Z}_N$. Let $\mathbf{z}_b^{(j)}$ be the j-th* *column of* $\mathbf{Z}_b$*,* $j = 1, \ldots, n$*. We have that* $\mathbf{z}_b^{(j)} = (N-1)\mathbf{1}_N - \mathbf{z}_{N/2+b}^{(j)}$ *for* $b = 0, 1, \ldots, N/2 - 1$*.*

Lemma 3. *Let* N *be odd,* $\mathbf{Z}_b$ *be as in Lemma 2 and* $\mathbf{z}_b^{(j)}$ *be its* j*-th column. For* $j = 1, \ldots, n$*, we have the following:*

(i) There exists an element $b^\star \in \mathbb{Z}_N$ *such that* $\mathbf{z}_{b^\star}^{(j)} = (N-1)\mathbf{1}_N - \mathbf{z}_{b^\star}^{(n+1-j)}$*. If* $(N-1)/2$ *is even,* $b^\star = (N-1)/4$*. Otherwise,* $b^\star = (3N-1)/4$*.*

(ii) For $b \neq b^\star$ *and* $b' = (N-1)/2 - b \pmod{N}$*,* $\mathbf{z}_b^{(j)} = (N-1)\mathbf{1}_N - \mathbf{z}_{b'}^{(n+1-j)}$*.*

Lemmas 1 and 2 imply that, when the number of runs is even, we only have to evaluate the LHDs $\mathbf{Z}_b$ obtained using the first half of the linear permutations, because the other LHDs have similar distance matrices. We state this formally below.

Theorem 1. *Let* N *be even and* $\mathbf{Z}_b$ *be as in Lemma 2. The distance matrices of* $\mathbf{Z}_b$ *and* $\mathbf{Z}_{N/2+b}$ *are the same for* $b = 0, \ldots, N/2 - 1$*.*

When the number of runs is odd, Lemmas 1 and 3 imply the next result.

Theorem 2. *For* N *an odd number, let* $\mathbf{Z}_b$*,* $b^\star$*,* b *and* b' *be as in Lemma 3. The distance matrix of* $\mathbf{Z}_{b^\star}$ *has* $n/2$ *repeated columns. Moreover, for each of the* $(N-1)/2$ *pairs* (b, b')*, the distance matrices of* $\mathbf{Z}_b$ *and* $\mathbf{Z}_{b'}$ *are the same up to column permutations.*

Table 1: L_1-distance of $\mathbf{Z}_b$ for different values of b in Example 1.

b	0	1	2	3	4	5	6	7	8	9	10
$d^1(\mathbf{Z}_b)$	10	39	31	31	39	10	28	34	30	34	28

LHDs with distance matrices that are the same up to column permutations have the same L_q-distance. From Theorem 2, we can evaluate only one LHD, say $\mathbf{Z}_b$, for each pair of linear permutations given by b and b', in addition to $\mathbf{Z}_{b^\star}$. The next example illustrates Lemma 3 and Theorem 2.

Example 1. We consider $N = 11$ where $\phi(11) = 10$ and $H = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. The GLP set $\mathbf{X}$ is an 11×10 LHD with elements $x_{i,j} = ij \pmod{11}$ for $i = 1, \ldots, 11$, and $j = 1, \ldots, 10$. For $b = 0, \ldots, 10$, we obtain $\mathbf{X}_b = \mathbf{X} + b \pmod{11}$ and $\mathbf{Z}_b = W(\mathbf{X}_b)$. Theorem 2 implies that there are five pairs $(b,\ b')$, for which the distance matrices of $\mathbf{Z}_b$ and $\mathbf{Z}_{b'}$ are the same up to column permutations. To illustrate this, Table 1 shows the L_1-distances of the 10 LHDs $\mathbf{Z}_b$. We see that $\mathbf{Z}_b$ and $\mathbf{Z}_{b'}$ have the same L_1-distance for $(b, b') = (0, 5), (1, 4), (2, 3), (6, 10)$ and $(7, 9)$. All these pairs satisfy $b' = (11-1)/2-b \pmod{11}$. Since $(N-1)/2 = 5$ is odd, Lemma 3(i) implies that $b^\star = (3N-1)/4 = 8$. Table 1 shows that this is the case since $\mathbf{Z}_8$ has an L_1-distance of 30, which is different from the other designs. Note that the value of $b^\star$ does not necessarily result in the best $\mathbf{Z}_b$.

Lemmas 1, 2 and 3 are relevant to our IP algorithm.

3.3 Strengths and limitations

For N a prime number and $n = N - 1$, the method of Wang et al. (2018) outperforms that of Xiao and Xu (2017), which constructs LHDs using the arrays of Costas (1984). Moreover, with few exceptions, it outperforms the SA algorithm of Ba et al. (2015) for $n = \phi(N)$ and $7 \leq N \leq 30$, in terms of the L_1-distance. When N is a prime, Wang et al. (2018) gives a formula to obtain the linear permutation that creates the best N-run $(N - 1)$-factor LHD in terms of the L_1-distance. This LHD is asymptotically optimal in terms of the maximin distance criterion as the ratio of its L_1-distance and the theoretical optimum converges to 1 for large N. Moreover, its average absolute correlation between two columns is smaller than $2/(N - 2)$. So, the larger the run size, the smaller its average absolute correlation.

Despite these attractive features, the method of Wang et al. (2018) has two limitations. First, it is unknown if it can generate good LHDs with a prime number of runs and fewer than $N - 1$ factors, in terms of the L_1-distance. Second, when N is not a prime, the largest number of factors of an LHD obtained by this method is $\phi(N) < N - 1$. For example, if N is 20, 24 or 30, the maximum number of factors is eight. So, LHDs with these run sizes and more than eight factors cannot be constructed with the method. Our IP algorithm, which we introduce next, overcomes these limitations.

4. Integer programming algorithm

We first briefly review integer programming. Next, we present the candidate set, problem formulation and implementation of the IP algorithm. We end the section with numerical experiments to assess its performance.

4.1 Background

Integer programming (IP) is an optimization method to determine the values of a set of discrete or continuous decision variables so as to optimize a linear objective function, while satisfying a set of linear constraints (Wolsey, 2020). To use IP in practice, we need a problem formulation and an optimization solver to find its optimal solution. A problem formulation has the following general form:

$$\max_{\mathbf{x}} \quad \mathbf{c}^T\mathbf{x} \quad \text{subject to} \tag{4.1a}$$

$$\mathbf{G}\mathbf{x} = \mathbf{b}, \;\; \mathbf{H}\mathbf{x} \leq \mathbf{d}, \;\; \mathbf{x} \geq \mathbf{0}_n, \tag{4.1b}$$

$$x_i \in \mathbb{Z}, \;\; \forall i \in \mathcal{J}, \tag{4.1c}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ is an $n \times 1$ vector of decision variables, $\mathbf{c}$ is an $n \times 1$ vector, $\mathbf{G}$ is an $m_1 \times n$ matrix, $\mathbf{H}$ is an $m_2 \times n$ matrix, $\mathbf{b}$ is an $m_1 \times 1$ vector, $\mathbf{d}$ is an $m_2 \times 1$ vector, $\mathbf{0}_n$ is the $n \times 1$ vector of zeros, and $\mathcal{J}$ is a nonempty set of indices. If $\mathcal{J} = \{1, 2, \ldots, n\}$, the problem is called the integer linear programming problem. Otherwise, it is called the mixed-integer linear programming (MILP) problem.

Commercial optimization solvers such as Gurobi, CPLEX and MOSEK can solve the problem formulation in (4.1a)–(4.1c). To this end, they use a branch-and-bound algorithm (Wolsey, 2020, ch. 7) which conducts a systematic exploration of the solution space through an enumeration tree. The nodes of the tree are subproblems of the problem in (4.1a)–(4.1c) that result from branching on the integer variables. Using bounds for the objective function's value of the subproblems, the branch-and-bound algorithm prunes the branches (and thus the nodes) of the tree. In this way, the algorithm avoids the exploration of all feasible solutions and speeds up the computations. To further increase the computational performance, the solvers use other state-of-the-art optimization techniques such as disjunctive programming for branching rules, primal heuristics, linear optimization methods, cutting plane theory, pre-processing techniques and symmetry breaking methods (Jünger et al., 2010).

During the optimization routine, the solvers provide both feasible solutions and bounds for the objective function's optimal value of the problem in (4.1a)–(4.1c). As a solver progresses toward the optimal solution, the bounds improve and provide an increasingly better guarantee of optimality, which is especially useful if the solver is stopped before it converges to the global optimum. This feature is not shared by algorithms developed from metaheuristics, which do not provide certificates of optimality of their solutions. IP has been successfully used to solve many optimization problems such as the bus and driver scheduling problem (Kang et al., 2019), the multi-trip vehicle routing problem (Neira et al., 2020), and the generalized traveling salesman problem (Yuan et al., 2021).

To the best of our knowledge, there are only two IP-based approaches for constructing LHDs that optimize the maximin distance criterion. Van Dam et al. (2007) propose an MILP problem to find LHDs that maximize the L_1- and L_∞-distances. However, their approach is limited to LHDs with two factors only. Van Dam et al. (2009) show an MILP problem to obtain bounds on the L_1-, L_2- and L_∞-distances of LHDs with more than two factors. A core component of their problem is a candidate set of permutations of the elements in $\mathbb{Z}_N$. More specifically, this set comprises $N!/2$ elements of the *full* set of permutations of the elements in $\mathbb{Z}_N$. The problem also involves integer variables, one for each column in the candidate set. A major limitation of their approach is that it is computationally demanding and often infeasible to solve, which prevents it from being practically relevant. For example, to construct LHDs with 12 runs or more, the MILP problem has at least 200 million integer decision variables!

4.2 The candidate set

The initial candidate set $\mathbf{C}$ that we consider is constructed by concatenating LHDs obtained from GLP sets, linear permutations and the Williams' transformation. More specifically, we first consider $\mathbf{C} = [\mathbf{Z}_0, \mathbf{Z}_1, \cdots, \mathbf{Z}_{N-1}]$ with $\mathbf{Z}_b$ as in Section 3.2. This candidate set allows the IP algorithm to inherit the strengths of the LHDs of Wang et al. (2018). However, from Lemmas 1, 2 and 3, this set has pairs of fully correlated columns, which is undesirable because they imply factors whose linear effects are fully aliased. To overcome this issue, we remove one column from each pair of fully correlated columns from the candidate set. Therefore, when the number of runs N is even, the final candidate set we use to construct LHDs is

$$\mathbf{C} = [\mathbf{Z}_0, \mathbf{Z}_1, \cdots, \mathbf{Z}_{N/2-1}]. \tag{4.2}$$

This candidate set has $nN/2$ columns with $n = \phi(N)$.

When N is odd, the final candidate set depends on whether $(N-1)/2$ is even or odd, because this defines the structures of (b, b') and the value of $b^\star$ in Lemma 3. To define this set, we need an additional notation. Let $\mathbf{Y}_b$ be the matrix involving the first $n/2$ columns of $\mathbf{Z}_b$ and $g(N) = (N-1)/2 + 1$. The final candidate set we use to construct LHDs with N odd is as follows:

- If $(N-1)/2$ is even,

$$\mathbf{C} = [\mathbf{Z}_0, \mathbf{Z}_1, \cdots, \mathbf{Z}_{b^\star-1}, \mathbf{Z}_{g(N)}, \mathbf{Z}_{g(N)+1}, \cdots, \mathbf{Z}_w, \mathbf{Y}_{b^\star}], \tag{4.3}$$

  where $b^\star = (N-1)/4$ and $w = \lfloor (3N-1)/4 \rfloor$.

- If $(N-1)/2$ is odd,

$$\mathbf{C} = [\mathbf{Z}_0, \mathbf{Z}_1, \cdots, \mathbf{Z}_w, \mathbf{Z}_{g(N)}, \mathbf{Z}_{g(N)+1}, \cdots, \mathbf{Z}_{b^\star-1}, \mathbf{Y}_{b^\star}], \tag{4.4}$$

  where $b^\star = (3N-1)/4$ and $w = \lfloor (N-1)/4 \rfloor$.

In either case, the set has $n(N-1)/2+n/2 = nN/2$ columns with $n = \phi(N)$.

The next result shows that, when N is an odd prime, the candidate set is of the highest-quality in terms of the L_1-distance.

Theorem 3. *If N is an odd prime, the candidate set $\mathbf{C}$ in (4.3) and (4.4) is a maximin L_1-distance LHD with N runs, $N(N-1)/2$ factors and an L_1-distance equal to $N(N^2-1)/6$.*

Example 2. Consider a simple case with $N = 5$, $\phi(5) = 4$ and $g(5) = 3$. Let $\mathbf{X}$ be the 5×4 GLP set, $\mathbf{X}_b = \mathbf{X} + b \pmod 5$, and $\mathbf{Z}_b = W(\mathbf{X}_b)$ be

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

for $b = 0, \ldots, 4$, respectively. Since $(N-1)/2 = 2$ is even, we have that $b^\star = 1$ and $w = 3$. Indeed, the first and second columns of $\mathbf{Z}_1$ are fully correlated with the fourth and third columns, respectively. This is because the elements in the first and second columns equal four minus the elements in the fourth and third columns, respectively. The first two columns of $\mathbf{Z}_1$ then form the 5×2 matrix $\mathbf{Y}_1$. Using a similar argument, we see that columns one, two, three and four of $\mathbf{Z}_0$ are fully correlated with columns four, three, two and one, respectively, of $\mathbf{Z}_2$. The same is true for $\mathbf{Z}_3$ and $\mathbf{Z}_4$. The final candidate set then is

$$\mathbf{C} = \begin{pmatrix} 2 & 4 & 3 & 1 & 1 & 0 & 2 & 4 & 4 & 3 \\ 4 & 1 & 2 & 3 & 0 & 4 & 1 & 2 & 3 & 0 \\ 3 & 2 & 1 & 4 & 2 & 1 & 4 & 0 & 1 & 4 \\ 1 & 3 & 4 & 2 & 4 & 2 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 3 & 3 & 3 & 3 & 2 & 2 \end{pmatrix}.$$

Since N is a prime number, Theorem 3 implies that $\mathbf{C}$ is a maximin L_1-distance LHD with five runs and 10 factors. Indeed, this design has an L_1-distance of 20 which equals the upper bound of Van Dam et al. (2009).

4.3 Problem formulation

For each column in the candidate set $\mathbf{C}$, we define a binary variable y_u. The variable y_u equals one if and only if the u-th column of $\mathbf{C}$ is in the LHD. For our problem formulation, the relevant element of the candidate set is its distance matrix. Consider the $r \times p$ L_1-distance matrix $\mathbf{A}_1$ of $\mathbf{C}$, with $r = N(N-1)/2$ and $p = N\phi(N)/2$. The IP problem formulation to construct an N-run k-factor LHD that maximizes the L_1-distance is:

$$\max_{\mathbf{y},\, t} \quad t \quad \text{subject to} \tag{4.5a}$$

$$\mathbf{1}_p^T \mathbf{y} = k, \quad \mathbf{A}_1 \mathbf{y} \geq t\mathbf{1}_r, \tag{4.5b}$$

$$t \in \mathbb{Z}^+, \quad y_u \in \{0, 1\}, \quad u = 1, \ldots, p. \tag{4.5c}$$

This problem formulation has p binary decision variables contained within $\mathbf{y} = (y_1, y_2, \ldots, y_p)^T$, an integer decision variable t, and $r + 1$ linear constraints contained within (4.5b). It is straightforward to recast this formulation in the form of the general IP formulation in (4.1a)–(4.1c).

The linear objective function in (4.5a) is in terms of the decision variable t only. The first constraint in (4.5b) implies that the final LHD has exactly k columns while the other constraints ensure that the L_1-distances between any two distinct rows of the LHD, given by $\mathbf{A}_1\mathbf{y}$, must be larger than or equal to t. Maximizing t then maximizes the minimum L_1-distance between two rows of the LHD. The constraints in (4.5c) ensure that t is a positive integer and the variables y_u are binary.

The rows of $\mathbf{A}_1$ define the constraints within (4.5b). Ideally, this matrix has no repeated rows. Otherwise, some constraints appear more than once in the problem formulation. These repetitions are thus redundant. The next result shows a candidate set whose distance matrix has repeated rows.

Theorem 4. *For N even, the distance matrix of the candidate set in* (4.2) *has $\frac{N}{2}(\frac{N}{2}-1)$ pairs of repeated rows.*

In this case, we thus recommend removing one row from each pair of repeated rows in the distance matrix, before using the problem formulation. So, when N is even, the number of constraints within (4.5b) is $(N/2)^2+1$.

The problem formulation in (4.5a)–(4.5c) is similar in spirit to that of Van Dam et al. (2009). However, it has $N\phi(N)/2$ binary decision variables instead of $N!/2$ integer variables as that from these authors. This is because our problem formulation is tailored to an attractive candidate set, whose columns can be either included once or excluded from the LHD.

After solving the problem formulation (4.5a)–(4.5c) to optimality, the output is the vector $\mathbf{y}$ where the nonzero y_u values indicate the columns of $\mathbf{C}$ that are in the N-run k-factor LHD, that maximizes the L_1-distance. This LHD is optimal among all k-factor subsets of $\mathbf{C}$. In principle, we can obtain N-run LHDs with a number of factors as large as $N\phi(N)/2$, which is the size of the candidate set. However, we restrict to LHDs with up to $N-1$ factors because they are more relevant in practice. Note that LHDs that optimize the general L_q-distance can be obtained by replacing $\mathbf{A}_1$ with

$\mathbf{A}_q$ in (4.5b), for a positive integer q.

4.4 Implementation in Gurobi

To solve the IP problem formulation, we use the solver Gurobi v.9.1.1. We use the default settings for all the tuning parameters of the solver except for the `TimeLimit` parameter, which controls the maximum time allowed for the optimization. To keep all the experiments in this article within computational reach, we set `TimeLimit` to 300 seconds. All our numerical experiments were carried out at the computer cluster of the Department of Statistics at UCLA. The cluster has 256 GB of RAM and 48 cores with an Intel(R) Xeon(R) Platinum 8160 CPU with 2.10 GHz.

The Gurobi solver reports information on the current progress of the optimization, the most relevant of which is the relative gap. This gap equals $(b-o)/o$, where o is the objective function value of the current best solution and b the best upper bound of the objective function value found so far. If the solver is stopped prematurely, a relative gap larger than zero indicates that the solver did not prove the optimality of the best solution found. A relative gap of zero means that the solver found the optimal solution.

4.5 Computational results and comparisons

We discuss numerical experiments to compare the IP algorithm with algebraic methods and benchmark algorithms in the literature. Supplementary Section S2 shows additional experiments to validate the components of the IP algorithm. More specifically, Section S2.1 demonstrates that our candidate set embeds attractive LHDs in terms of the maximin distance criterion. Section S.2.2 shows that the IP problem generates better LHDs than those obtained by selecting columns at random from the candidate set.

We consider the design problems in Table 2, which we obtained from Wang et al. (2018). They involve LHDs with seven to 30 runs and four to 28 factors. For these design problems, Wang et al. (2018) report the L_1-distances of LHDs obtained by their method, Zhou and Xu (2015), Xiao and Xu (2017), and the SA algorithm of Ba et al. (2015). The SA algorithm was executed 100 times with its default parameters values, and the best design in terms of the L_1-distance was reported. This algorithm is implemented in the 'SLHD' package in the statistical software R. For completeness, Table 2 reproduces the L_1-distances in Wang et al. (2018) for these methods.

As an additional benchmark algorithm, we consider the genetic algorithm of Liefvendahl and Stocki (2006) because this and the SA algorithm are the best algorithms available to construct good LHDs in terms of the maximin distance criterion; see Zhou and Xu (2015), Xiao and Xu (2018), and Wang et al. (2021). For the genetic algorithm, we use its recommended parameter settings and its implementation in the 'LHD' package in R. To limit its computing time, the R implementation has a tuning parameter called the number of generations, which we set to 500 following Wang et al. (2021). Table 2 includes the L_1-distances of the genetic algorithm.

Table 2 shows that the IP algorithm matches or improves upon the benchmark methods for most design problems. More specifically, for 16, 19, 20, 23, 25, 27 and 28 runs, the LHDs obtained by the IP algorithm outperform all benchmark designs in terms of the L_1-distance. For the other cases, our algorithm generated LHDs that have the same L_1-distance as the best benchmark designs, except for 9, 10, 15, 18, 24 and 29 runs; see Table 2. The 10-, 15-, 18- and 24-run LHDs obtained by the genetic algorithm have an L_1-distance that is one or two units larger than our designs. For 9 and 29 runs, the LHDs obtained by the SA algorithm and Wang et al. (2018), respectively, have a larger L_1-distance than our designs.

Except for LHDs with 19, 23, 25, 27 and 29 runs, the Gurobi solver certified that all LHDs constructed by the IP algorithm have the best possible L_1-distance, among those obtained from the candidate set in Section 4.2. Therefore, 9-, 10-, 15-, 18- and 24-run LHDs with L_1-distances larger than those in Table 2 cannot be obtained with our candidate sets. For the LHDs

Table 2: L_1-distances of N-run k-factor LHDs with $k = \phi(N)$.

N	k	IP	SA	GA	ZX	WXX	XX	N	k	IP	SA	GA	ZX	WXX	XX
7	6	**16**	15	15	13	**16**	14	19	18	**118**	108	110	106	115	106
8	4	**11**	**11**	10	8	10		20	8	**47**	43	46	32	42	
9	6	17	**18**	17	15	16		21	12	**77**	73	**77**	66	76	
10	4	11	11	**12**	8	11		22	10	**68**	61	64	60	**68**	
11	10	**39**	36	38	34	**39**	34	23	22	**172**	160	161	154	168	158
12	4	**13**	**13**	**13**	8	10		24	8	53	50	**54**	32	36	
13	12	**54**	52	52	**54**	52	48	25	20	**163**	153	153	147	162	
14	6	**24**	23	**24**	22	**24**		26	12	**98**	87	91	84	**98**	
15	8	36	35	**37**	29	36		27	18	**157**	145	147	135	156	
16	8	**43**	37	39	32	36		28	12	**104**	92	97	72	94	
17	16	**94**	86	89	84	**94**	86	29	28	270	254	254	250	**274**	250
18	6	28	28	**30**	18	28		30	8	**63**	57	**63**	40	61	

IP: IP algorithm; SA: simulated annealing algorithm; GA: genetic algorithm; ZX: Zhou and Xu (2015); WXX: Wang et al. (2018); XX: Xiao and Xu (2017). The largest L_1-distance for each design problem is shown in bold font.

with 19, 23, 25, 27 and 29 runs, the upper bounds on the L_1-distance are 119, 176, 168, 162 and 280, respectively. This means that the relative gaps between the best solutions and upper bounds range from 1.69% to 3.32% in these cases. Therefore, better LHDs may be obtained if we increase the computing time of the solver.

Here, we concentrated on the construction of N-run LHDs with $\phi(N)$ factors, where N ranges from seven to 30. However, our IP algorithm can construct LHDs with more or fewer factors than $\phi(N)$. For instance, it can generate an LHD with up to $N-1$ factors for each value of N in Table 2.

5. A modified IP algorithm for constructing large designs

The IP problem in Section 4.3 is a cardinality-constrained optimization problem which is NP-hard (Bienstock, 1996). However, our previous computational experiments show that the Gurobi solver can find good or even optimal solutions for design problems with up to 30 runs and up to 29 factors, within five minutes. This renders our IP algorithm as computationally feasible for constructing LHDs of small and moderate sizes.

For larger-sized LHDs, our algorithm inevitably suffers from the complexity of the IP problem. To overcome this issue, we reduce the candidate set and include an extra step in the IP algorithm. We now present these modifications and a numerical evaluation of their performance.

5.1 A reduced candidate set and the leave-one-out method

The reduced candidate set, which we denote as $\mathbf{D}$, is the best N-run $(N-1)$-factor LHD with N a prime number from the method of Wang et al. (2018). More specifically, $\mathbf{D}$ is constructed using the $N \times (N-1)$ GLP set, the Williams' transformation, and the linear permutation that results in the best LHD in terms of the L_1-distance; see Wang et al. (2018) for a formula to obtain this permutation. We choose this candidate set because it allows us to generate LHDs with up to $N-1$ factors. Moreover, $\mathbf{D}$ is asymptotically optimal in terms of the maximin distance criterion and has small correlations between its columns.

Using $\mathbf{D}$ instead of the full candidate set in (4.3) or (4.4) results in a problem formulation with the same number of constraints but $p = N - 1$ binary decision variables; see (4.5a)–(4.5c). Although the resulting problem formulation is still NP-hard, it has a smaller solution space than the original formulation which involves $N(N-1)/2$ binary decision variables. Compared to the latter, the smaller solution space of the former allows the Gurobi solver to generate LHDs with large N and k values.

With the reduced candidate set, we can construct N-run LHDs with a number of factors $k \leq N - 1$, where N is a prime number. To generate LHDs with fewer than N runs, we sequentially apply the leave-one-out method (Fang and Wang, 1981). Let M be the run size of the desired LHD and assume that $M < N$. First, we generate N reduced designs by removing each point in the N-run k-factor LHD. Next, we convert each reduced design into an LHD by rearranging its entries. To this end, we rearrange the entries of a reduced design column by column. If the entry with value $x \in \mathbb{Z}_N$ is removed from a column, the entries larger than x are decreased by one. After that, we evaluate the N resulting LHDs with $N-1$ runs and k factors, and select the best one in terms of the L_1-distance. To find an LHD with $N-2$ runs, we repeat the whole procedure using the best $(N-1)$-run LHD as a start. Using the newly obtained smaller LHD, we repeat the procedure again to generate an $(N-3)$-run LHD and so on, until we obtain an M-run LHD.

5.2 Computational performance

We compare the modified IP algorithm with the SA algorithm of Ba et al. (2015) and the genetic algorithm of Liefvendahl and Stocki (2006), for constructing large LHDs. The computational setup of the algorithms is the same as before. However, preliminary experiments (not shown here) revealed the benchmark algorithms are computationally demanding for large numbers of runs or factors. Therefore, we imposed an additional stopping rule: we set their maximum computing time to that of the Gurobi solver. For each design problem, we executed the SA algorithm 100 times and reported the best LHD obtained among all iterations that were completed within 300 seconds. Similarly, we use 500 generations of the genetic algorithm and recorded the best LHD obtained within this time frame.

5.2.1 Large LHDs with a prime number of runs

We begin with design problems involving 31, 47, 71 and 97 runs, all of which are prime numbers. For each run size N, we consider five numbers of factors: $\lfloor (N-1)/4 \rfloor$, $\lfloor (N-1)/3 \rfloor$, $\lfloor (N-1)/2 \rfloor$, $\lfloor 2(N-1)/3 \rfloor$, and $\lfloor 3(N-1)/4 \rfloor$. We chose these numbers of factors because they range from small to large relative to the run size. These design problems allow us to assess the quality of LHDs obtained from subsets of columns of the N-run $(N-1)$-factor LHDs of Wang et al. (2018), with N a prime number.

Table 3 shows the L_1-distances of LHDs obtained by the modified IP (M-IP), SA and genetic algorithms. The M-IP algorithm outperforms the SA and genetic algorithms in 13 of the 20 design problems in the table. Our algorithm is generally better than the others for large run sizes.

In Table 3, all 31-run LHDs of our algorithm are optimal among LHDs obtained from subsets of columns of the initial 31-run 30-factor LHD. For all other combinations of numbers of runs and numbers of factors in the table, the Gurobi solver did not finish the search for the optimal LHD within 300 seconds. For 47, 71 and 97 runs, the relative gaps given by the solver ranged from 1.0% to 5.8%, 5.1% to 22.4%, and 5.4% to 20.7%, respectively.

Table 3: L_1-distances of N-run k-factor LHDs with N a prime number.

N	k	M-IP	SA	GA	N	k	M-IP	SA	GA
31	7	46	49	**54**	71	17	**303**	299	300
	10	77	80	**85**		23	**448**	437	422
	15	129	131	**134**		35	**725**	710	670
	20	**187**	182	185		46	**999**	971	890
	22	207	204	**208**		52	**1149**	1104	1029
47	11	120	122	**134**	97	24	**613**	600	538
	15	184	183	**190**		32	846	**847**	779
	23	**310**	306	303		48	**1386**	1338	1190
	30	**425**	418	405		64	**1904**	1843	1630
	34	**492**	476	462		72	**2199**	2099	1872

M-IP: modified IP algorithm; SA: simulated annealing algorithm; GA: genetic algorithm.

5.2.2 Large LHDs with general run sizes

Additionally, we consider the design problems in Table 4, which involve 44 to 96 runs and 23 to 72 factors. In these cases, the number of runs is not a prime and so, the M-IP algorithm uses the leave-one-out method. The starting designs for this step were the LHDs in Table 3.

Table 4 shows the L_1-distances obtained by the M-IP, SA and genetic algorithms. For all design problems, the M-IP algorithm outperforms the benchmark algorithms. The performance of our algorithm was particularly excellent for LHDs with 68 runs or more, and 46 factors or more. This

Table 4: L_1-distances of N-run k-factor LHDs with N not a prime number.

N	k	M-IP	SA	GA	N	k	M-IP	SA	GA
44	23	**292**	291	286	70	35	**718**	705	654
	30	**400**	390	387		46	**987**	951	844
	34	**463**	447	434		52	**1134**	1089	1014
45	23	**298**	296	295	94	48	**1349**	1306	1161
	30	**409**	399	392		64	**1851**	1790	1591
	34	**473**	458	437		72	**2132**	2037	1846
46	23	**304**	302	298	95	48	**1361**	1313	1170
	30	**416**	409	402		64	**1870**	1817	1637
	34	**482**	471	445		72	**2154**	2054	1814
68	35	**699**	687	624	96	48	**1374**	1348	1172
	46	**959**	935	862		64	**1889**	1827	1609
	52	**1104**	1060	987		72	**2176**	2080	1818
69	35	**709**	693	650					
	46	**972**	941	861					
	52	**1120**	1071	981					

is because the L_1-distances of these designs are larger than those of the benchmark algorithms by at least 24 units.

5.2.3 LHDs with a large number of runs relative to the number of factors

We also investigate the performance of the M-IP algorithm for constructing

Table 5: L_1-distances of N-run k-factor LHDs with $k = \lfloor N/10 \rfloor$.

N	k	M-IP	SA	GA	N	k	M-IP	SA	GA
71	7	83	89	**94**	101	10	**203**	191	187
73		87	91	**96**	103		**211**	196	190
79		89	95	**100**	107		**211**	207	188
83	8	**130**	123	124	109		**212**	206	188
89		**133**	127	128	113	11	**244**	242	223
97	9	161	**163**	155					

LHDs with run sizes that are considerably larger relative to the number of factors. For illustrative purposes, we consider a prime number of runs N ranging from 71 to 113. Following Loeppky et al. (2009), we use a number of factors $k = \lfloor N/10 \rfloor$ for each of the 11 values of N.

Table 5 shows that, for all design problems with 8, 10 and 11 factors, the M-IP algorithm produce better LHDs than the benchmark algorithms in terms of the L_1-distance. Therefore, the M-IP algorithm has the potential to generate attractive LHDs with a large number of runs relative to the number of factors; particularly, when the number of factors is at least eight.

6. Concluding remarks

In this article, we introduced the IP algorithm to construct LHDs that optimize the maximin distance criterion, as measured by the L_1-distance. The algorithm is rooted in integer programming and uses a candidate set of attractive columns to generate the designs. We generated this set from LHDs obtained by Wang et al. (2018) and used novel theoretical results to avoid fully correlated columns in the set. Remarkably, when the run size is a prime, the candidate set is in itself a maximin L_1-distance LHD. Using numerical experiments, we showed that the IP algorithm is computationally-effective for small and moderate design problems. For larger-sized design problems, we modified the algorithm by reducing the candidate set and using the leave-one-out method to obtain LHDs with any run size. We demonstrated that the modified IP algorithm outperforms the benchmark algorithms in 47 of our 58 design problems. Our algorithm is particularly effective for constructing LHDs with around 100 runs, which supports the idea that it will outperform the benchmark algorithms for larger run sizes.

For 29 runs and 28 factors, the IP algorithm did not generate a better LHD than Wang et al. (2018). This may be because the time allowed for the optimization by the Gurobi solver was not sufficient. Indeed, additional computations revealed that, after four hours, the solver found an LHD with an L_1-distance of 275, which is larger than all benchmark designs in Table 2. We therefore recommend to complete the optimization of the IP algorithm.

In principle, we could use the modified IP algorithm to generate small-and moderately-sized LHDs. As a proof of concept, we used this alternative algorithm to construct LHDs for the design problems in Table 2, involving a run size that is not a prime number. However, the resulting LHDs were not better than those of the standard IP algorithm. We therefore recommend the modified IP algorithm for situations in which the standard IP algorithm is computationally infeasible. We also recommend to complete the optimization of the modified IP algorithm. Only in situations in which short computing times are desired, we may suggest to impose a user-specified maximum computing time for the Guorbi solver, as we did here. In any case, our numerical experiments show that the standard and modified IP algorithms can generally obtain good LHDs within five minutes, subject to a similar computer hardware and software as ours.

The IP algorithm is general since it works for any candidate set and, with simple modifications, can construct LHDs that maximize other distances such as the L_2-distance. Although we did not generate LHDs that optimize this distance, the Cauchy-Schwarz inequality shows that the L_1-distance is a lower bound of the L_2-distance. Therefore, our LHDs would tend to perform well in terms of the L_2-distance too.

Using a modified Williams' transformation, Wang et al. (2018) construct maximin L_1-distance LHDs with N runs and N factors, subject to

$2N + 1$ being a prime number. To test whether our IP algorithm can obtain these designs, we constructed LHDs with N equal to five, six, eight, nine, 11, 14 and 15. The resulting LHDs were optimal in terms of the L_1-distance among all comparable LHDs obtained from subsets of columns of the candidate set. They, however, did not match the L_1-distance of the maximin distance LHDs. So, these optimal LHDs are not embedded in the candidate sets we considered. This calls for alternative candidate sets for the IP algorithm, but we leave this quest for future research.

Another topic for future research is to extend the problem formulation in Section 4.3, so as to both optimize the maximin distance criterion and minimize the correlations between the columns of the LHDs. To this end, we may use the problem formulations of Harris et al. (1995) and Hernandez et al. (2012) to construct LHDs that minimize these correlations. In contrast with the heuristic algorithm of Joseph and Hung (2008), this multi-objective approach would provide certificates of optimality of the LHDs.

Supplementary Materials

The online supplementary materials include the proofs of the theoretical results, additional numerical experiments to validate the IP algorithm, and a Python implementation of its standard and modified versions.

Acknowledgements

The authors thank an Associate Editor and three reviewers for their helpful comments. The research of Vazquez was supported by the Flemish Fund for Scientific Research (FWO) through a Junior Postdoctoral Fellowship.

References

Achterberg, T. and R. Wunderling (2013). Mixed integer programming: Analyzing 12 years of progress. In M. Jünger and G. Reinelt (Eds.), *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*, pp. 449–481. Berlin, Heidelberg: Springer Berlin Heidelberg.

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

Bienstock, D. (1996). Computational study of a family of mixed-integer quadratic programming problems. *Mathematical Programming 74*, 121–140.

Bixby, R. (2012). A brief history of linear and mixed-integer programming computation. *Doumenta Mathematica. Extra Volume: Optimization Stories*, 107–121.

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

Costas, J. P. (1984). A study of a class of detection waveforms having nearly ideal range-doppler ambiguity properties. *Proceedings of the IEEE 72*, 996–1009.

Fang, K. T., R. Li, and A. Sudjianto (2006). *Design and Modeling for Computer Experiments*. Chapman & Hall/CRC Press, Boca Raton, FL.

Fang, K.-T., D. K. Lin, P. Winker, and Y. Zhang (2000). Uniform design: Theory and application. *Technometrics 42*, 237–248.

Fang, K. T. and Y. Wang (1981). A note on uniform distribution and experiment design. *Chinese Science Bulletin 26*, 485–489.

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

Harris, C. M., K. L. Hoffman, and L. A. Yarrow (1995). Using integer programming techniques for the solution of an experimental design problem. *Annals of Operations Research 58*, 243–260.

Hernandez, A. S., T. W. Lucas, and M. Carlyle (2012). Constructing nearly orthogonal Latin hypercubes for any nonsaturated run-variable combination. *ACM Transactions on Modeling and Computer Simulation 22*, 1–17.

Houston, D. X., S. Ferreira, J. S. Collofello, D. C. Montgomery, G. T. Mackulak, and D. L. Shunk (2001). Behavioral characterization: Finding and using the influential factors in software process simulation models. *The Journal of Systems and Software 59*, 259–270.

Jin, R., W. Chen, and A. Sudjianto (2005). An efficient algorithm for constructing optimal design of computer experiments. *Journal of Statistical Planning and Inference 134*, 268–287.

Johnson, M., L. Moore, and D. Ylvisaker (1990). Minimax and maximin distance designs. *Journal of Statistical Planning and Inference 26*, 131–148.

Joseph, V. R., T. Dasgupta, R. Tuo, and C. F. J. Wu (2015). Sequential exploration of complex surfaces using minimum energy designs. *Technometrics 57*, 64–74.

Joseph, V. R. and Y. Hung (2008). Orthogonal-maximin distance Latin hypercube designs. *Statistica Sinica 18*, 171–186.

Jünger, M., T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey (Eds.) (2010). *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art.* Springer.

Kang, L., S. Chen, and Q. Meng (2019). Bus and driver scheduling with mealtime windows for a single public bus route. *Transportation Research Part C: Emerging Technologies 101*, 145–160.

Liefvendahl, M. and R. Stocki (2006). A study on algorithms for optimization of Latin hypercubes. *Journal of Statistical Planning and Inference 136*, 3231–3247.

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

McKay, M. D. (1995). Evaluating prediction uncertainty. Los Alamos National Laboratory,

NUREGICR-6311 (LA-12915-MS).

Moon, H., A. Dean, and T. Santner (2011). Algorithms for generating maximin Latin hypercube and orthogonal designs. *Journal of Statistical Theory and Practice 5*, 81–98.

Morris, M. D. and T. J. Mitchell (1995). Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference 43*, 381–402.

Neira, D. A., M. M. Aguayo, R. De la Fuente, and M. A. Klapp (2020). New compact integer programming formulations for the multi-trip vehicle routing problem with time windows. *Computers & Industrial Engineering 144*, 106399.

Pronzato, L. and W. G. Müller (2012). Design of computer experiments: space filling and beyond. *Statistics and Computing 22*, 681–701.

Santner, T. J., B. J. Williams, and W. I. Notz (2018). *The Design and Analysis of Computer Experiments* (2nd ed.). Springer, New York.

Van Dam, E. R., B. Husslage, and D. d. Hertog (2007). Maximin Latin hypercube designs in two dimensions. *Operations Research 55*, 158–169.

Van Dam, E. R., G. Rennen, and B. Husslage (2009). Bounds for maximin Latin hypercube designs. *Operations Research 57*, 595–608.

Wang, H., Q. Xiao, and A. Mandal (2021). Musings about constructions of efficient Latin hypercube designs with flexible run-sizes. arXiv:2010.09154 [stat.ME].

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

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

Wolsey, L. A. (2020). *Integer Programming* (2nd ed.). Wiley-Interscience publication.

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

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

Ye, K. Q., W. Li, and A. Sudjianto (2000). Algorithmic construction of optimal symmetric Latin hypercube designs. *Journal of Statistical Planning and Inference 90*, 145–159.

Yuan, Y., D. Cattaruzza, M. Oiger, C. Rousselot, and F. Semet (2021). Mixed integer programming formulations for the generalized traveling salesman problem with time windows. *4OR 19*, 571–592.

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

Department of Industrial Engineering, University of Arkansas, Arkansas, 72701, U.S.A.

E-mail: alanv@uark.edu

Department of Statistics, University of California, Los Angeles, California, 90095, U.S.A.

E-mail: hqxu@stat.ucla.edu