GENERATING A MAXIMALLY SPACED SET OF BINS TO FILL FOR HIGH-DIMENSIONAL SPACE-FILLING LATIN HYPERCUBE SAMPLING

***Keith R. Dalbey*[1] & *George N. Karystinos*[2,*]**

[1] *Department of Optimization and Uncertainty Quantification, Sandia National Laboratories, Albuquerque, New Mexico 87123, USA*

[2] *Department of Electronic and Computer Engineering, Technical University of Crete, Kounoupidiana, Chania, 73100, Greece*

Original Manuscript Submitted: 10/15/2010; Final Draft Received: 12/12/2010

In the literature, space-filling Latin hypercube sample designs typically are generated by optimizing some criteria such as maximizing the minimum distance between points or minimizing discrepancy. However, such methods are time consuming and frequently produce designs that are highly regular, which can bias results. A fast way to generate irregular space-filling Latin hypercube sample designs is to randomly distribute the sample points to a pre-selected set of well-spaced bins. Such designs are said to be "binning optimal" and are shown to be irregular. Specifically, Fourier analysis reveals regular patterns in the multi-dimensional spacing of points for the Sobol sequence but not for Binning optimal symmetric Latin hypercube sampling. For $M = 2^r \leq 8$ dimensions and $N = 2^s \geq 2M$ points, where r and s are non-negative integers, simple patterns can be used to create a list of maximally spaced bins. Good Latin hypercube sample designs for non-power of two dimensions can be generated by discarding excess dimensions. Since the octants/bins containing the $2M$ end points of an "orientation" (a rotated set of orthogonal axes) are maximally spaced, the process of generating the list of octants simplifies to finding a list of maximally spaced orientations. Even with this simplification, the "patterns" for maximally spaced bins in $M \geq 16$ dimensions are not so simple. In this paper, we use group theory to generate $2^M/(2M)$ disjoint orientations, and present an algorithm to sort these into maximally spaced order. Conceptually, the procedure works for arbitrarily large numbers of dimensions. However, memory requirements currently preclude even listing the $2^M/(2M)$ orientation leaders for $M \geq 32$ dimensions. In anticipation of overcoming this obstacle, we outline a variant of the sorting algorithm with a low memory requirement for use in $M \geq 32$ dimensions.

KEY WORDS: *uncertainty quantification, Monte Carlo, Latin hypercube sampling, space-filling, computational design, high-dimensional methods, regularity detection*

1. INTRODUCTION

Models (i.e., simulators) are frequently used to make predictions about the performance of physical systems. However, the correct input values for these models are often uncertain, and making accurate predictions of system behavior requires that this uncertainty be propagated to the models' output. Randomly drawing samples from the distributions of uncertain inputs is one of the oldest [1], most robust, and universally applicable methods of uncertainty quantification (UQ). Because the error in a sample mean computed from a Monte Carlo sampling (MCS) design with N points scales as $N^{-1/2}$, one million simulations will generally be needed for three significant figures of accuracy. When the simulator is sufficiently expensive, running one million simulations is unfeasible and faster methods are required.

[*] Correspond to George N. Karystinos, E-mail: karystinos@telecom.tuc.gr, URL: http://www.telecom.tuc.gr/∼karystinos/

Some UQ methods, such as stochastic collocation (SC) [2], exploit smoothness in the model's output by using a set of samples to construct a fast surrogate for the computationally expensive simulator. Unfortunately, SC requires that all members of a specific deterministic set of simulations successfully execute. Other methods, such as Kriging and Bayesian emulation, have less dependence on the system's smoothness and are more robust in that they do not require a specific set of simulations. However, the quality of the sample design does affect the accuracy of the fitted surface. High-quality random sample designs are also desirable because they are less susceptible to bias error than deterministic sample designs.

Latin hypercube sampling (LHS) and jittered sampling (JS) both achieve better convergence than standard MCS by using stratification to obtain a more uniform distribution of samples, although LHS and JS use different stratification strategies. LHS is space filling in the one-dimensional projections but not in the full M-dimensional space. JS is space filling in the full M-dimensional space but not in the one-dimensional projections. When the input distributions are non-uniform, an appropriate sample design can often be obtained by mapping (frequently through the cumulative distribution functions) and/or weighting the sample points.

Greater accuracy with fewer samples can be achieved by combining both stratification strategies. Consequently, generating space-filling Latin hypercubes has long been an active area of research [3–17]. Typically, space-filling Latin hypercube sample designs are generated by optimizing some criteria. Examples of space-filling criteria include: maximizing the minimum (maximin) distance between points, minimizing the maximum (minimax) distance between points, maximizing the entropy, minimizing the integrated mean square error (IMSE), minimizing the Audze-Eglais potential energy, and minimizing discrepancy. Discrepancy is appealing as a criterion because of the "Koksma-Hlawka-like inequality." It states that the error in a sample mean is bounded above by the product of the sample design's L_p discrepancy and the function's L_q variance, where $p^{-1} + q^{-1} = 1$. However, generating space-filling Latin hypercubes through optimization is time consuming and frequently produces sample designs that are highly regular, which can bias results.

In a recent work, Dalbey and Karystinos [18] presented an $\mathcal{O}[N\log(N)]$ fast algorithm to generate irregular space-filling Latin hypercube sample designs. Their binning optimal symmetric Latin hypercube sampling (BOSLHS) algorithm randomly distributes sample points to a set of well-spaced bins. The resulting sample designs had low centered and wrap-around L_2 discrepancies (smaller is better), high coverage (larger is better), low correlations between different input dimensions (smaller is better), and a low "t" quality rating (smaller is better) when the designs are considered to be tms-nets. The centered L_2 discrepancy of several sampling methods are plotted for $M = 4$ and $M = 8$ dimensions in Fig. 1. BOSLHS has the lowest centered L_2 discrepancy of all random methods compared.

Let C^M be the M-dimensional unit hypercube $[0,1]^M$, $b \geq 2$ be a prime number, and P be defined as

$$P = \mathrm{ceil}\left(\frac{\log_b(N)}{M}\right)$$

Then, a design is "binning optimal" with respect to base b if two conditions are met.

1. When C^M is divided into a uniform grid of cube bins with volume $\mathrm{vol} = b^{-PM}$, no bin contains more than one point.

2. When C^M is divided into a uniform grid of cube bins with volume $\mathrm{vol} = b^{-(P-1)M}$ (i.e., one generation larger), every bin contains the same number of points.

Binning optimality ensures that in the limit of an infinite number of samples the sample design will include every point in the input space. This means that binning optimal designs are space filling.

A sample design's degree, if any, of binning non-optimality can be determined in $\mathcal{O}[N\log(N)] + \mathcal{O}(NM)$ operations. This is accomplished by computing a bin identifier, henceforth "id," for each point as its index into the Morton or "Z" space-filling curve, followed by quick sorting the Z-curve ids and tallying their frequency of occurrence. The sorted Z-curve ids can also be used to detect regularity (i.e., cyclic patterns) in the multi-dimensional spacing of sample points. Figures 2–4 show sample designs with $M = 4$ and $N = 256$ points and the corresponding amplitude versus frequency plots (found by taking a fast Fourier transform of the difference of sequential sorted Z-curve ids)

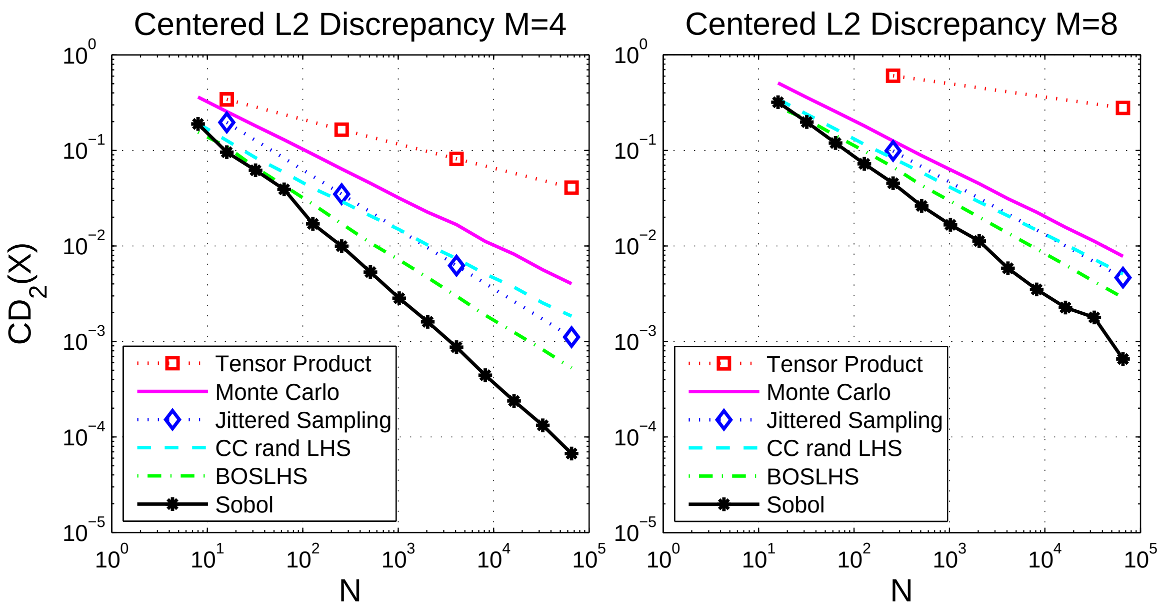

FIG. 1: Centered L_2 discrepancy (lower is better) as a function of number of points, N, in $M = 4$ and $M = 8$ dimensions for tensor product sampling, MCS, JS, cell-centered Latin hypercube sampling with randomly paired dimensions, BOSLHS, and the Sobol sequence. Except for tensor product sampling and the Sobol sequence, which are both completely deterministic, the lines plot the average discrepancy for 40 randomly generated designs.

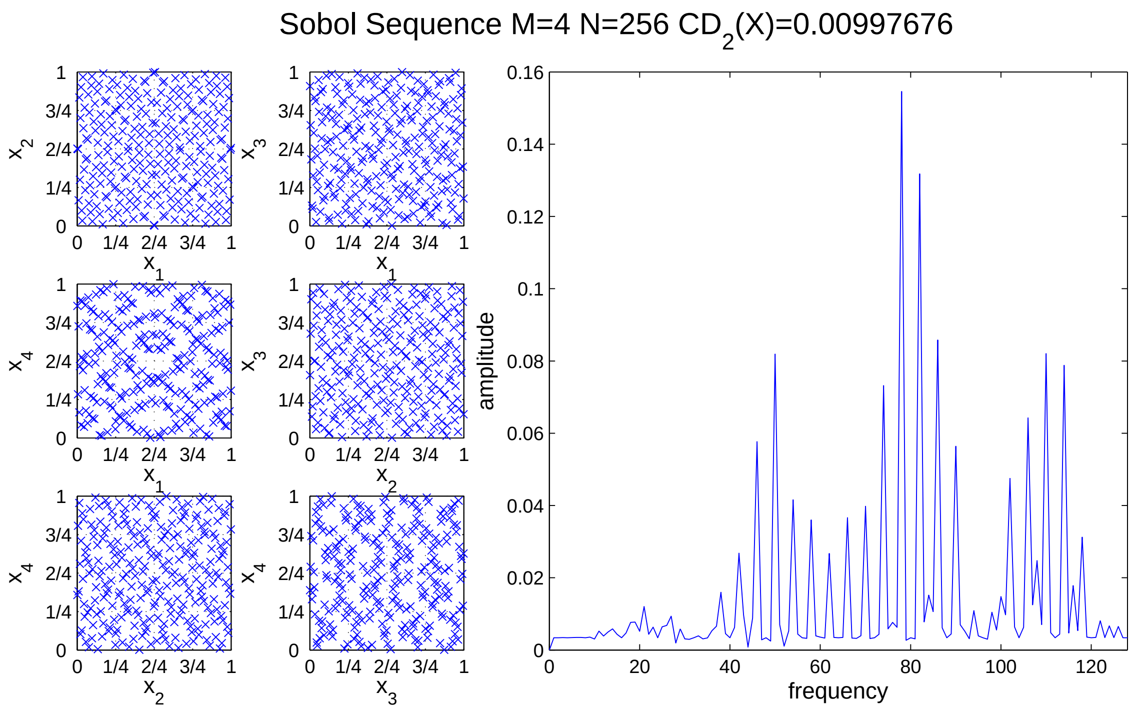

FIG. 2: Plots of all combinations of 2 out of $M = 4$ dimensions and the amplitude versus frequency for a Sobol sequence design with $N = 256$ points. Although it has low discrepancy, the Sobol sequence is highly regular, which can bias results.

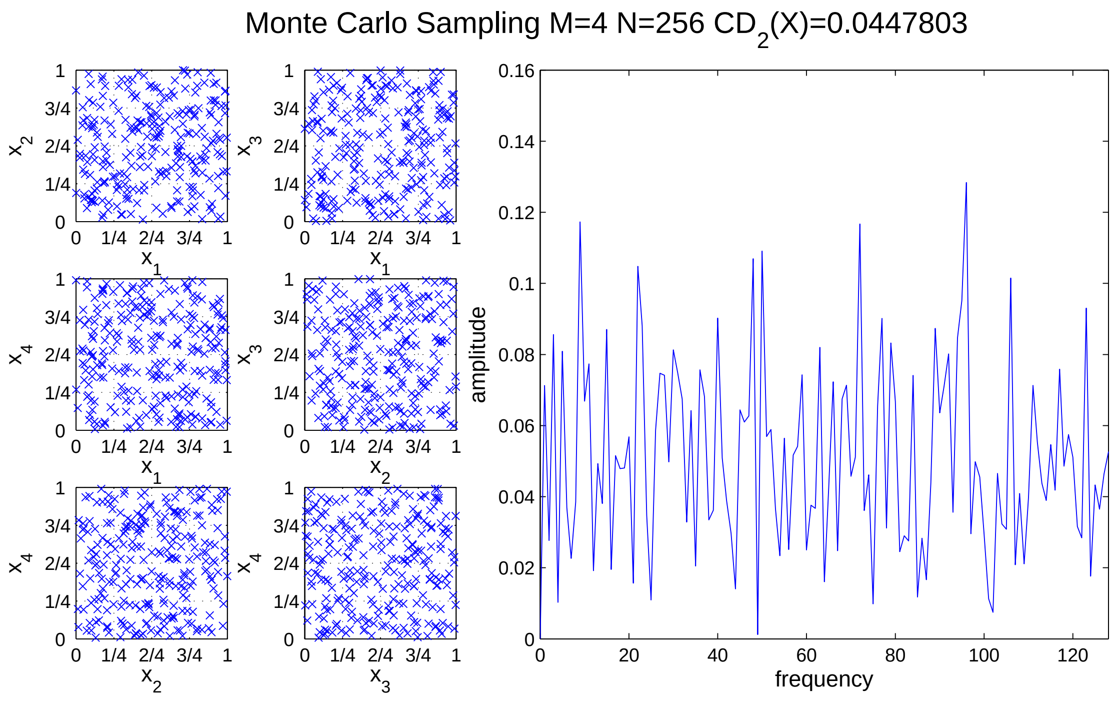

FIG. 3: Plots of all combinations of 2 out of $M = 4$ dimensions and the amplitude versus frequency for a Monte Carlo sample design with $N = 256$ points. The amplitude versus frequency plot is noisy because Monte Carlo sampling is completely irregular.

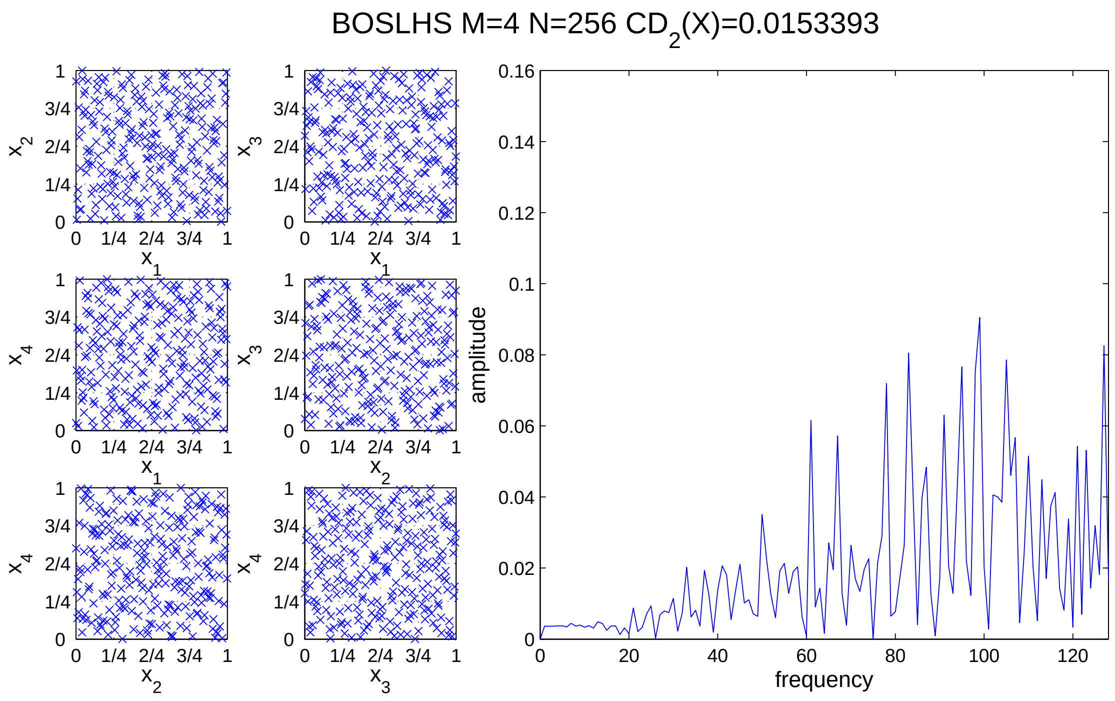

FIG. 4: Plots of all combinations of 2 out of $M = 4$ dimensions and the amplitude versus frequency for a BOSLHS design with $N = 256$ points. BOSLHS is not visibly regular and the amplitude versus frequency plot confirms that observation.

for the Sobol sequence, MCS, and BOSLHS designs, respectively. The Sobol sequence is clearly regular; MCS and BOSLHS are not.

Note that binning optimality is a rather weak sort of optimality. It verifies that there are the "right" number of points in bins with an edge length of b^{-P} or larger. It does not verify that the subgrid-scale location of each point is optimal in any sense. Said another way, binning optimality does not imply the optimal spacing of bins that contain points for bins of edge length b^{-P} or smaller. However, Z-curve ids can be used to engineer maximal spacing of bins of edge length 2^{-P} into the LHS construction procedure.

For $M = 2^r \leq 8$ dimensions and $N = 2^s \geq 2M$ points, where r and s are non-negative integers, simple patterns can be used to create a list of maximally spaced bins. Good LHS designs for non-power of two dimensions can be generated by discarding excess dimensions; the degree of binning non-optimality can be used to quickly compare the various candidate combinations of dimensions to retain. Since the octants/bins containing the $2M$ end points of an "orientation" (defined as a rotated set of orthogonal axes) are maximally spaced, creating the list of octants simplifies to finding a list of maximally spaced orientations. However, even with this simplification, the "patterns" for maximally spaced bins in $M \geq 16$ dimension are not so simple. In this paper, we present a procedure for generating a list of octants in maximally spaced order for higher dimensions.

In Section 2, we use group theory to generate $2^M/(2M)$ disjoint orientations. In Section 3, we demonstrate how to sort the list of orientations into maximally spaced order. Conceptually, our procedure works for an arbitrarily large number of dimensions. However, memory requirements preclude even listing the $2^M/(2M)$ orientation leaders for $M \geq 32$ dimensions.

2. GENERATING THE LIST OF DISJOINT ORIENTATIONS

For any M equal to a power of 2, a Hadamard matrix $\check{\mathbf{H}}_M$ is a $M \times M$ matrix with elements in $\{\pm 1\}$ and the property that any row differs from any other row in exactly $M/2$ positions; hence, its rows are orthogonal to each other. If, in addition, the elements of the first row of the matrix are all equal to 1, then it is a normalized Hadamard matrix.

Normalized Hadamard matrices can be iteratively constructed through Sylvester construction. For $M = 2$, the normalized Hadamard matrix is

$$\check{\mathbf{H}}_2 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix}. \tag{1}$$

Furthermore, from $\check{\mathbf{H}}_M$, we can generate the Hadamard matrix $\check{\mathbf{H}}_{2M}$ according to the relation

$$\check{\mathbf{H}}_{2M} = \begin{bmatrix} \check{\mathbf{H}}_M & \check{\mathbf{H}}_M \\ \check{\mathbf{H}}_M & -\check{\mathbf{H}}_M \end{bmatrix}. \tag{2}$$

For example, for $M = 4$, we obtain

$$\check{\mathbf{H}}_4 = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \end{bmatrix}. \tag{3}$$

Let M be a power of 2 and $\check{\mathbf{H}}$ the corresponding normalized Hadamard matrix. The corresponding orientation contains $2M$ octants (considering both endpoints of each axis) that are represented by the rows of matrix

$$\check{\mathbf{C}} = \begin{bmatrix} \check{\mathbf{H}} \\ -\check{\mathbf{H}} \end{bmatrix}. \tag{4}$$

Let $\check{\mathbf{c}}_1, \check{\mathbf{c}}_2, \ldots, \check{\mathbf{c}}_{2M} \in \{\pm 1\}^M$ be the $2M$ rows of $\check{\mathbf{C}}$. For example, for $M = 4$, the orientation matrix is

$$\check{\mathbf{C}} = \begin{bmatrix} +1 & +1 & +1 & +1 \\ +1 & -1 & +1 & -1 \\ +1 & +1 & -1 & -1 \\ +1 & -1 & -1 & +1 \\ -1 & -1 & -1 & -1 \\ -1 & +1 & -1 & +1 \\ -1 & -1 & +1 & +1 \\ -1 & +1 & +1 & -1 \end{bmatrix} \tag{5}$$

and $\check{\mathbf{c}}_1$, $\check{\mathbf{c}}_2$, ..., $\check{\mathbf{c}}_8$ are the eight rows of $\check{\mathbf{C}}$.

Our objective is to find the sequence of vectors $\{\check{\mathbf{e}}_k\}_{k=1}^{(2^M/2M)}$, where $\check{\mathbf{e}}_k \in \{\pm 1\}^M$, $k = 1, 2, \ldots, (2^M/2M)$, such that the corresponding sequence of orientation matrices $\{\check{\mathbf{W}}^{(k)}\}_{k=1}^{(2^M/2M)}$ that are formed by

$$\check{\mathbf{W}}^{(k)} = \begin{bmatrix} \check{\mathbf{c}}_1 \odot \check{\mathbf{e}}_k \\ \check{\mathbf{c}}_2 \odot \check{\mathbf{e}}_k \\ \vdots \\ \check{\mathbf{c}}_{2M} \odot \check{\mathbf{e}}_k \end{bmatrix}, \quad k = 1, 2, \ldots, \frac{2^M}{2M}, \tag{6}$$

do not share common axes. In Eq. 6, operator $\odot$ stands for the Hadamard (that is, entrywise) product. Note that, by construction, no orientation matrix $\check{\mathbf{W}}^{(k)}$, $k = 1, 2, \ldots, (2^M/2M)$, contains identical rows.

To significantly reduce the complexity of our following algorithmic developments, it will be convenient to use algebraic operations by mapping the alphabet $\{+1, -1\}$ to the binary field $\mathbb{F}_2$, consisting of $\{0, 1\}$ with modulo-2 addition and modulo-2 multiplication ($0 + 0 = 1 + 1 = 0$; $0 + 1 = 1$; $0 \cdot 0 = 1 \cdot 0 = 0$; $1 \cdot 1 = 1$).[1] We use the standard mapping

$$\{\pm 1\} : \begin{array}{ccc} +1 & \longleftrightarrow & 0 \\ -1 & \longleftrightarrow & 1 \end{array} : \mathbb{F}_2. \tag{7}$$

Under this mapping, the multiplication (of integers) over $\{\pm 1\}$ is equivalently represented as modulo-2 addition over $\mathbb{F}_2$. For example, the multiplication $[-1\ +1\ +1\ +1] \odot [+1\ -1\ -1\ +1] = [-1\ -1\ -1\ +1]$ over $\{\pm 1\}$ is represented as modulo-2 addition $[1\ 0\ 0\ 0] + [0\ 1\ 1\ 0] = [1\ 1\ 1\ 0]$ over $\mathbb{F}_2$.

Let $\mathbf{H}$ and $\mathbf{C} = \left[\begin{smallmatrix}\mathbf{H} \\ \overline{\mathbf{H}}\end{smallmatrix}\right]$, where $\overline{\mathbf{H}}$ is the complement of $\mathbf{H}$, be the matrices with elements in $\mathbb{F}_2$ that correspond—under the above mapping—to the normalized Hadamard matrix $\check{\mathbf{H}}$ and the corresponding orientation matrix $\check{\mathbf{C}}$, respectively. We call them the normalized Hadamard $\mathbb{F}_2$ matrix and the orientation $\mathbb{F}_2$ matrix, respectively. Observe that from $\mathbf{H}_M$ we can generate $\mathbf{H}_{2M}$ according to the relation

$$\mathbf{H}_{2M} = \begin{bmatrix} \mathbf{H}_M & \mathbf{H}_M \\ \mathbf{H}_M & \overline{\mathbf{H}}_M \end{bmatrix}. \tag{8}$$

In addition, let $\mathbf{c}_1$, $\mathbf{c}_2$, ..., $\mathbf{c}_{2M} \in \mathbb{F}_2^M$ be the $2M$ rows of $\mathbf{C}$. For example, for $M = 4$, we obtain the normalized Hadamard $\mathbb{F}_2$ matrix

$$\mathbf{H}_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \tag{9}$$

and the corresponding orientation $\mathbb{F}_2$ matrix

[1] In simple words, a field is a set of elements in which we can perform addition, subtraction, multiplication, and division without leaving the set. Addition and multiplication must satisfy the commutative, associative, and distributed laws.

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

whose eight rows are denoted by $\mathbf{c}_1$, $\mathbf{c}_2$, ..., $\mathbf{c}_8$.

Then, we can rewrite our objective as follows. Find the sequence of vectors $\{\mathbf{e}_k\}_{k=1}^{(2^M/2M)}$, where $\mathbf{e}_k \in \mathbb{F}_2^M$, $k = 1, 2, \ldots, (2^M/2M)$, such that the corresponding sequence of orientation $\mathbb{F}_2$ matrices $\{\mathbf{W}^{(k)}\}_{k=1}^{(2^M/2M)}$ that are formed (using modulo-2 addition) by

$$\mathbf{W}^{(k)} = \begin{bmatrix} \mathbf{c}_1 + \mathbf{e}_k \\ \mathbf{c}_2 + \mathbf{e}_k \\ \vdots \\ \mathbf{c}_{2M} + \mathbf{e}_k \end{bmatrix}, \quad k = 1, 2, \ldots, \frac{2^M}{2M}, \tag{11}$$

do not share common axes. Note that, by construction, no orientation $\mathbb{F}_2$ matrix $\mathbf{W}^{(k)}$, $k = 1, 2, \ldots, (2^M/2M)$ contains identical rows.

It can be proven that the set $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{2M}\}$, which consists of the row vectors of $\mathbf{C}$, forms a $\log(2M)$-dimensional subspace of the vector space of all the 2^M M-tuples over the field $\mathbb{F}_2$. In fact, the latter is true since the modulo-2 addition of any two rows of $\mathbf{C}$ is also a row of $\mathbf{C}$, that is, $\mathbf{c}_i + \mathbf{c}_j \in \mathbf{C}$, $\forall\, \mathbf{c}_i, \mathbf{c}_j \in \mathbf{C}$, as it can be easily checked from the definition of $\mathbf{C}$. A method to partition the 2^M M-tuples into $(2^M/2M)$ orientation $\mathbb{F}_2$ matrices [or, equivalently, to find the sequence of vectors $\{\mathbf{e}_k\}_{k=1}^{(2^M/2M)}$] such that no orientation contains identical rows and no pair of orientations share a common axis is described below. The partition is based on the fact that the rows of $\mathbf{C}$ form a $\log(2M)$-dimensional subspace of the vector space of all the 2^M M-tuples over the field $\mathbb{F}_2$.

We utilize the procedure presented in [19]. First, we use the $2M$ rows of $\mathbf{C}$ to form a super row with the all-zero vector $\mathbf{c}_1$ (which we also call $\mathbf{e}_1$) as the first (leftmost) element. We choose a vector from the remaining $2^M - 2M$ vectors in $\mathbb{F}_2^M$, call it $\mathbf{e}_2$, and place it under the zero vector $\mathbf{c}_1$. Next, we form a second super row by (modulo-2) adding $\mathbf{e}_2$ to each vector $\mathbf{c}_i$ in the first row and placing the sum $\mathbf{e}_2 + \mathbf{c}_i$ under $\mathbf{c}_i$. After the second super row has been completed, an unused vector from the remaining $2^M - 4M$ M-tuples is chosen, called $\mathbf{e}_3$, and placed under $\mathbf{c}_1$. Then, a third super row is formed by (modulo-2) adding $\mathbf{e}_3$ to each vector $\mathbf{c}_i$ in the first super row and placing $\mathbf{e}_3 + \mathbf{c}_i$ under $\mathbf{c}_i$. This process is continued until all 2^M vectors in $\mathbb{F}_2^M$ have been used. The result is an array of super rows and super columns, in the standard array of $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{2M}\}$ as follows:

$$\begin{bmatrix} \mathbf{c}_1 = \mathbf{0} & \mathbf{c}_2 & \cdots & \mathbf{c}_i & \cdots & \mathbf{c}_{2M} \\ \mathbf{e}_2 & \mathbf{e}_2 + \mathbf{c}_2 & \cdots & \mathbf{e}_2 + \mathbf{c}_i & \cdots & \mathbf{e}_2 + \mathbf{c}_{2M} \\ \vdots & \vdots & & \vdots & & \vdots \\ \mathbf{e}_k & \mathbf{e}_k + \mathbf{c}_2 & \cdots & \mathbf{e}_k + \mathbf{c}_i & \cdots & \mathbf{e}_k + \mathbf{c}_{2M} \\ \vdots & \vdots & & \vdots & & \vdots \\ \mathbf{e}_{(2^M/2M)} & \mathbf{e}_{(2^M/2M)} + \mathbf{c}_2 & \cdots & \mathbf{e}_{(2^M/2M)} + \mathbf{c}_i & \cdots & \mathbf{e}_{(2^M/2M)} + \mathbf{c}_{2M} \end{bmatrix} \tag{12}$$

In the context of error-correction coding in digital communications, this array is called a *standard array* of the linear block code $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{2M}\}$. More details about the construction of a standard array can be found in [19].

A standard array has the following important properties:

(i) The sum of any two vectors in the same super row is a row of $\mathbf{C}$ (that is, a vector in the first super row).
Proof: Consider two vectors of super row k, say $\mathbf{e}_k + \mathbf{c}_i$ and $\mathbf{e}_k + \mathbf{c}_j$. Then, $(\mathbf{e}_k + \mathbf{c}_i) + (\mathbf{e}_k + \mathbf{c}_j) = \mathbf{c}_i + \mathbf{c}_j$

which is a row in $\mathbf{C}$ because the rows of $\mathbf{C}$ form a $\log(2M)$-dimensional subspace of the vector space of all the 2^M M-tuples over the field $\mathbb{F}_2$, as mentioned above.

(ii) No two vectors in the same super row are identical.
Proof: We suppose that two vectors in the kth super row are identical, say $\mathbf{e}_k + \mathbf{c}_i = \mathbf{e}_k + \mathbf{c}_j$ with $i \neq j$. Then, $\mathbf{c}_i = \mathbf{c}_j \in \mathbf{C}$, which is impossible, since $i \neq j$ and, by construction, all rows of $\mathbf{C}$ are distinct.

(iii) Every vector appears in one and only one super row.
Proof: It is apparent that every vector appears at least once, since every vector is used at least once during the construction of the standard array. We have to show that no vector appears more than once. We begin by supposing that a vector appears in both the kth super row and the lth super row with $k < l$. Then, due to the construction of the standard array, this vector can be expressed as $\mathbf{e}_k + \mathbf{c}_i$ for some i and as $\mathbf{e}_l + \mathbf{c}_j$ for some j. Hence, $\mathbf{e}_k + \mathbf{c}_i = \mathbf{e}_l + \mathbf{c}_j$, which implies that $\mathbf{e}_l = \mathbf{e}_k + (\mathbf{c}_i + \mathbf{c}_j)$. Because $\mathbf{c}_i, \mathbf{c}_j \in \mathbf{C}$ and the rows of $\mathbf{C}$ form a $\log(2M)$-dimensional subspace, it is implied that $\mathbf{c}_i + \mathbf{c}_j$ is also a row of $\mathbf{C}$, say $\mathbf{c}_m$. Then, $\mathbf{e}_l = \mathbf{e}_k + \mathbf{c}_m$. This equality implies that vector $\mathbf{e}_l$ belongs to the kth super row of the array. If the latter statement held true, then $\mathbf{e}_l$ should not have been used at the beginning of a lower super row. However, $\mathbf{e}_l$ is indeed used at the beginning of the lth super row with $l > k$. Therefore, we arrive at a contradiction and conclude that no vector can appear more than once in the array.

From Properties (ii) and (iii), we observe that there are $2^M/2M$ disjoint super rows in the standard array and that each super row consists of $2M$ distinct vectors. Hence, the matrix

$$\mathbf{W}^{(k)} = \begin{bmatrix} \mathbf{e}_k \\ \mathbf{e}_k+\mathbf{c}_2 \\ \vdots \\ \mathbf{e}_k+\mathbf{c}_{2M} \end{bmatrix}$$

which consists of the vectors in the kth super row of the standard array is the kth orientation we are looking for where, by definition, $\mathbf{e}_k$ is the corresponding vector we are looking for. For example, for $M = 4$, the standard array becomes

$$\begin{bmatrix} 0000 & 0101 & 0011 & 0110 & 1111 & 1010 & 1100 & 1001 \\ 1000 & 1101 & 1011 & 1110 & 0111 & 0010 & 0100 & 0001 \end{bmatrix}$$

where $\mathbf{e}_1 = \mathbf{c}_1 = [0\ 0\ 0\ 0]$ and $\mathbf{e}_2 = [1\ 0\ 0\ 0]$. The two super rows of the array represent the two orientations we are looking for.

In the context of error-correction coding in digital communications, the $2^M/2M$ super rows of the standard array are called the *cosets* of the linear block code $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{2M}\}$ and the first vector $\mathbf{e}_k$ of each coset is called a *coset leader* (or coset representative). Apparently, any vector of a coset can be used as its leader. If another vector of a coset becomes its leader, then the vectors that constitute the coset do not change but are only permuted. Particularly, to minimize the probability of a channel decoding error, it can be proven that when the standard array is formed each coset leader should be chosen to be a vector of minimum number of ones from the remaining available vectors. It is implied that if coset leaders are chosen in this manner, then each coset leader has a minimum number of ones among the vectors of the coset.

Because the role of the standard array is critical for channel decoding in digital communications, there have been efficient implementations of the standard array of any linear block code. These implementations usually construct the set of coset leaders in the above manner to minimize channel decoding error probability. An example of such an implementation is MATLAB function syndtable, which accepts as input a parity-check matrix of the linear block code and returns a matrix whose rows represent the coset leaders from the code's standard array. The parity-check matrix is easily constructed from the generator matrix of the linear block code which, in turn, consists of exactly as many linearly independent elements of the code as its dimensionality.

In the context of our developments, the rows $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{2M}$ of the original orientation $\mathbb{F}_2$-matrix $\mathbf{C}$ form a $\log(2M)$-dimensional subspace (linear block code). We select exactly $\log(2M)$ linearly independent rows from $\mathbf{C}$ to construct the generator matrix of the code and (through standard Gaussian-elimination-based techniques) form the corresponding parity-check matrix of the code. Then, we call MATLAB function syndtable with the input parity-check matrix constructed above and obtain the $2^M/2M$ coset (or orientation) leaders $\mathbf{e}_1 = \mathbf{0}$, $\mathbf{e}_2, \ldots, \mathbf{e}_{(2^M/2M)}$. These are the vectors we are looking for. We convert the coset leaders into the $\{\pm 1\}$ alphabet, through the mapping in Eq. (7), to obtain $\check{\mathbf{e}}_1, \check{\mathbf{e}}_2, \ldots, \check{\mathbf{e}}_{(2^M/2M)}$. Finally, we construct the $2^M/2M$ orientation matrices through Eq. (6).

For example, for $M = 4$, from the eight rows $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_8$ of matrix $\mathbf{C}$ in Eq. (10), we select $\log(2M) = 3$ linearly independent rows, namely, $\mathbf{c}_8 = [1\ 0\ 0\ 1]$, $\mathbf{c}_2 = [0\ 1\ 0\ 1]$, and $\mathbf{c}_3 = [0\ 0\ 1\ 1]$, to form the generator matrix $\left[\begin{smallmatrix}1&0&0&1\\0&1&0&1\\0&0&1&1\end{smallmatrix}\right]$. From this, we form the parity-check matrix $[1\ 1\ 1\ 1]$, which is fed to function syndtable that, in turn, outputs matrix $\left[\begin{smallmatrix}0&0&0&0\\1&0&0&0\end{smallmatrix}\right]$ whose rows are the $2^M/2M = 2$ coset leaders we are looking for. We convert the coset leaders into the $\{\pm 1\}$ alphabet, through the mapping in Eq. (7), to obtain $\check{\mathbf{e}}_1 = [1\ 1\ 1\ 1]$ and $\check{\mathbf{e}}_2 = [-1\ 1\ 1\ 1]$ and—finally—construct the corresponding two orientation matrices $\check{\mathbf{W}}^{(1)}$ and $\check{\mathbf{W}}^{(2)}$ through Eq. (6).

3. SORTING THE LIST OF ORIENTATIONS INTO MAXIMALLY SPACED ORDER

In order to sort the orientations into maximally spaced order, we first need a criterion to measure distance. For that purpose, we define the dot product of two orientations $\check{\mathbf{W}}^{(i)}$ and $\check{\mathbf{W}}^{(j)}$ to be

$$\operatorname{dot}\left(\check{\mathbf{W}}^{(i)}, \check{\mathbf{W}}^{(j)}\right) = \max\left\langle \operatorname{mod}\left\{\operatorname{abs}\left[\check{\mathbf{W}}^{(i)}\left(\check{\mathbf{W}}^{(j)}\right)^T\right], M\right\}\right\rangle. \tag{13}$$

Note that this definition says that the dot product of an orientation with itself is zero, so it is not actually an inner product. We opted to use the informal name, “dot product,” instead of calling it a “semi-inner product,” since the operator does not satisfy the Cauchy-Schwarz inequality.

The smaller the dot product of two orientations, the greater the Hamming distance between the two closest octants in them. That minimum Hamming distance, $\delta\left(\check{\mathbf{W}}^{(i)}, \check{\mathbf{W}}^{(j)}\right)$, is given by

$$\delta\left(\check{\mathbf{W}}^{(i)}, \check{\mathbf{W}}^{(j)}\right) = \left[M - \operatorname{dot}\left(\check{\mathbf{W}}^{(i)}, \check{\mathbf{W}}^{(j)}\right)\right]/2.$$

In $M = 16$ dimensions, there are $2^M/(2M) = 2048$ orientations. This means that there are potentially ${}_{2048}\mathrm{C}_2 = 2,096,128$ orientation dot products that we need to compute, store, and use to sort the orientations. However, we can reduce this to two groups of 523,776 dot products.

As indicated in Section 2, the orientation leaders for $M = 4$ dimensions are

$$\begin{bmatrix} + & + & + & + \\ - & + & + & + \end{bmatrix}.$$

Likewise, for $M = 8$, the $2^M/(2M) = 16$ orientation leaders are

$$\begin{bmatrix} + & + & + & + & + & + & + & + \\ - & - & + & + & + & + & + & + \\ - & + & - & + & + & + & + & + \\ - & + & + & - & + & + & + & + \\ - & + & + & + & - & + & + & + \\ - & + & + & + & + & - & + & + \\ - & + & + & + & + & + & - & + \\ - & + & + & + & + & + & + & - \end{bmatrix} \quad \begin{bmatrix} - & + & + & + & + & + & + & + \\ + & - & + & + & + & + & + & + \\ + & + & - & + & + & + & + & + \\ + & + & + & - & + & + & + & + \\ + & + & + & + & - & + & + & + \\ + & + & + & + & + & - & + & + \\ + & + & + & + & + & + & - & + \\ + & + & + & + & + & + & + & - \end{bmatrix}.$$

Note that these can/have been divided into two groups. In the first group, all octants have an even number of negative bit signs. In the second, all octants have an odd number of negative bit signs. The minimum Hamming distance between nearest octants in the first and second groups is 1 bit sign. Within each group, the minimum Hamming distance between any two octants is 2 bit signs.

For $M = 16$ dimensions, we can likewise separate the total of 2048 orientations into 1024 even and 1024 odd orientations, and then calculate dot products for, and sort, the two groups independently. The even-odd separation is our “step 1 sort.” The dot products for the even orientations after the step 1 sort are shown in Fig. 5. Within the group of 1024 even (or odd) orientations the dot product can only have the following values $\{0, 4, 8, 12\}$; these correspond to minimum Hamming distances of $\{8, 6, 4, 2\}$ and the colors of {blue, aqua, yellowish-orange, and reddish-brown}, respectively. Recall that only the dot product of an orientation with itself is zero.

By analogy to the $M = 8$-dimensional case, we know that for $M = 16$ dimensions we can sort the 1024 orientations into 16 subgroups of 64 orientations such that the minimum Hamming distance between subgroups is 2 and the minimum Hamming distance within a subgroup is 4. That is our “step 2 sort.” The dot products for the even orientations after the step 2 sort are shown in Fig. 6. Note the introduction of blue + aqua + yellowish-orange (no reddish-brown) 64×64 “squares” along the diagonal.

For our “step 3 sort,” we sort each sub-group independently into eight sub-sub-groups of eight orientations such that the minimum Hamming distance within a sub-sub-group is 6. The dot products for the even orientations after the step 3 sort are shown in Fig. 7. Note the introduction of blue + aqua (no yellowish-orange or reddish-brown) 8×8 squares along the diagonal.

Before sorting the even (or odd) orientations, we first define a symmetric $2^M/(4M)$ by $2^M/(4M)$ dot product matrix $\mathbf{D}$ such that

$$D_{i,j} = \text{dot}\left(\check{\mathbf{W}}^{(i)}, \check{\mathbf{W}}^{(j)}\right) = \text{dot}\left(\check{\mathbf{W}}^{(j)}, \check{\mathbf{W}}^{(i)}\right). \tag{14}$$

This matrix has zeros on the diagonal.

3.1 The Step 2 Sort

Let d be the number of rows and columns of $\mathbf{D}$. For $i = 1 \to d$, sort the rows of the matrix $\mathbf{D}$ into ascending order based on the first i columns of $\mathbf{D}$. The second column breaks ties in the first; the third column breaks ties in the first and second, and so on. If all first i columns of two rows tie, which of the two comes first is unimportant at this point.

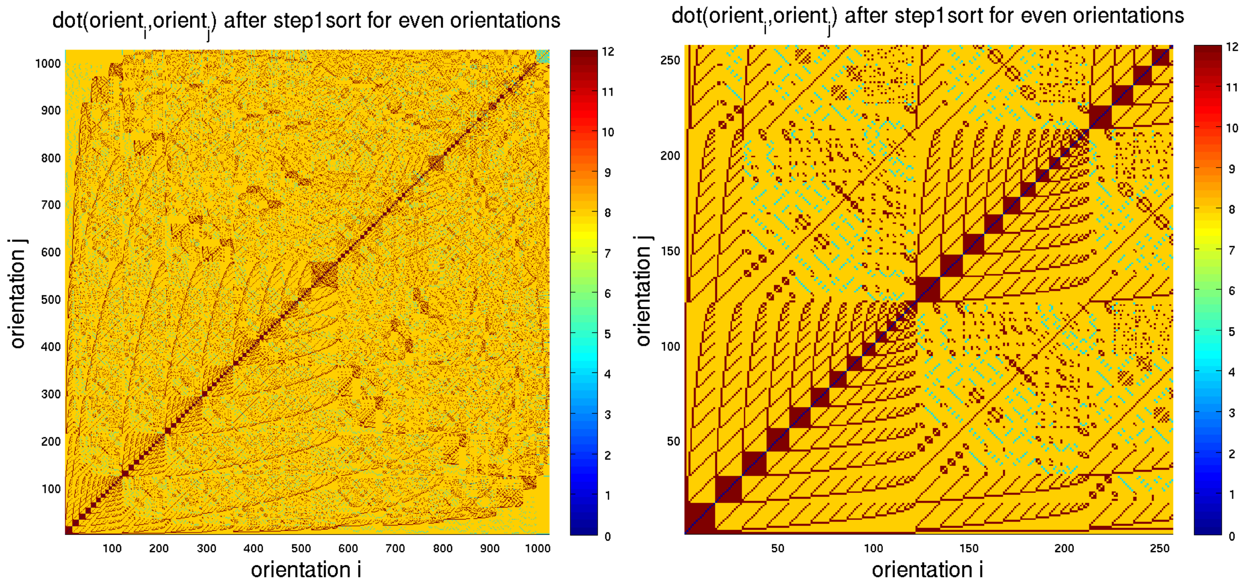

FIG. 5: Plots of the dot products of even orientations after the step 1 sort for $M = 16$ dimensions. The left subplot shows all 1024 even orientations; the right subplot shows only the first 256. The orientations are not in maximally spaced order after the step 1 sort but there is a minimum Hamming distance of 2 (dot product=12, reddish brown) between all even octants.

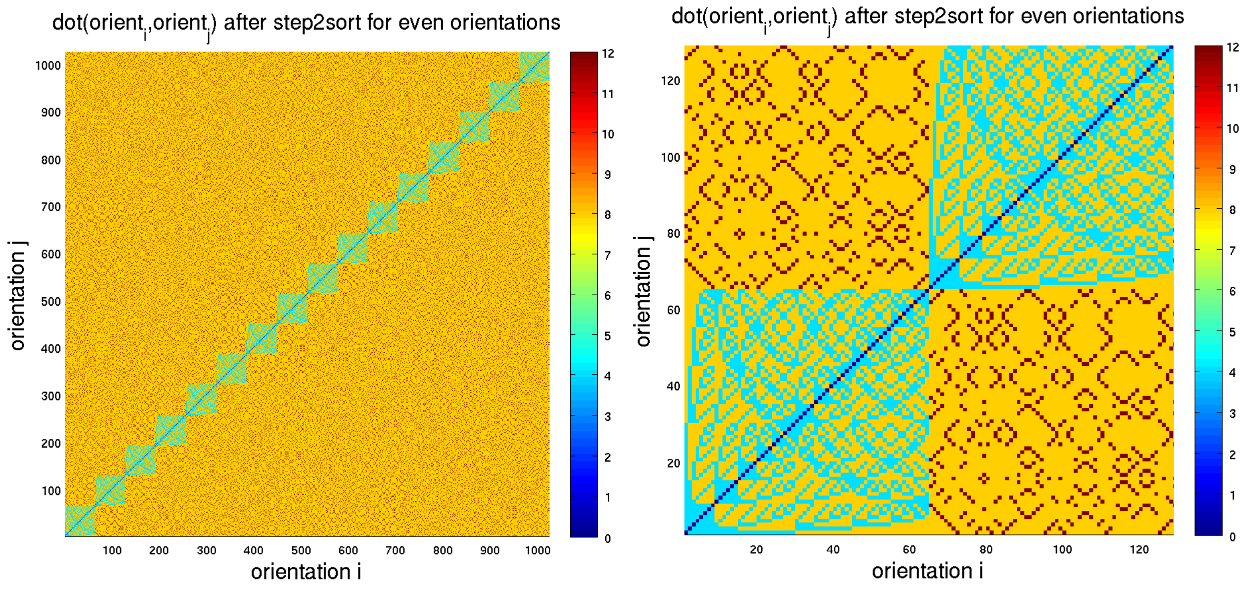

FIG. 6: Plots of the dot products of even orientations after the step 2 sort for $M = 16$ dimensions. The left subplot shows all 1024 even orientations; the right subplot shows only the first 128. The orientations are separated into subgroups with 64 members each. The minimum Hamming distance between octants within each subgroup is 4 bit signs (dot product=8, yellowish orange). The first eight orientations in a subgroup are separated by a minimum Hamming distance of 6 bit signs (dot product=4, aqua).

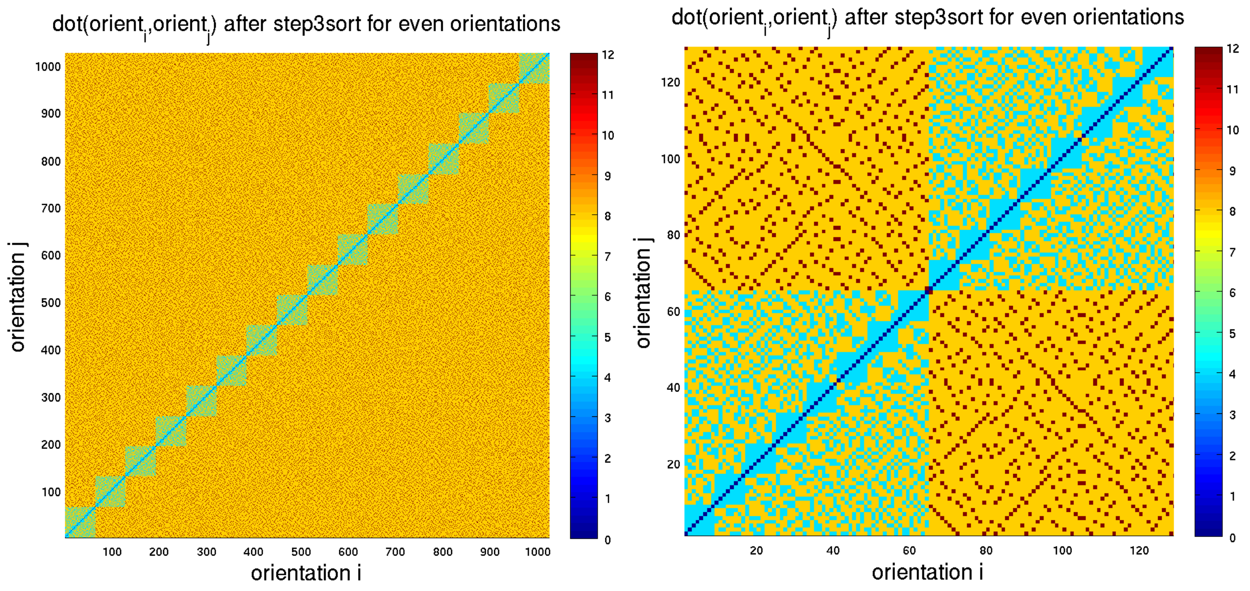

FIG. 7: Plots of the dot products of even orientations after the step 3 sort for $M = 16$ dimensions. The left subplot shows all 1024 even orientations; the right subplot shows only the first 128. Each subgroup of 64 orientations are separated into eight sub-sub-groups with eight members each. The minimum Hamming distance between octants within each sub-sub-group is 6 bit signs (dot product=4, aqua).

That will be dealt with during the step 3 sort. Immediately apply the same reordering to the columns of $\mathbf{D}$. Let k be the index of the last row for which all of the first i columns are less than or equal to $(3/4)M$. If $k <= i$ or $k = d$, stop the loop over i and apply the step 3 sort to the first k rows and columns of $\mathbf{D}$. Apply the same reordering of rows/columns $1 \rightarrow k$ of $\mathbf{D}$ to columns/rows $k+1$ through n of rows/columns $1 \rightarrow k$ of $\mathbf{D}$. Then apply the step 2 sort to the submatrix formed from rows and columns $k+1 \rightarrow d$ of $\mathbf{D}$. Then apply the same reordering of rows/columns $k+1 \rightarrow d$ of $\mathbf{D}$ to columns/rows $k+1 \rightarrow n$ of row/columns $1 \rightarrow k$ of $\mathbf{D}$. The step 2 sort can be implemented as a recursive function.

If it were possible to compute and store the orientation leaders for $M = 32$ dimensions, the first time that the above implementation of the step 2 sort would call itself is when $i = k = 2^{20} = 1,048,576$. The minimum Hamming distance separating octants in this set would be 4 bit signs. It is unlikely that a UQ application would require more than 2^{20} orientations ($N = 2^{26}$ samples), so a recursive call would not be needed. The problem of finding 2^{20} orientations separated by a minimum Hamming distance of 4 bit signs can be solved in a memory efficient way by implementing the step 2 sort as one column (of dot products) at a time procedure with a current tie-breaking-only sort. For each column, all orientations that are less than 4 bit signs away, would be eliminated from future consideration. It should also be possible, and faster (since a greater number of orientations would be eliminated from each column) to find orientations separated by a minimum Hamming distance of $M/4 = 8$ bit signs using the same one column at a time implementation of the step 2 sort. Although the step 2 sort would be expensive, it is a one time cost since the result could be stored for future reuse. All octants would be reachable by randomly generating a string of 32 bit signs and applying it to all leaders in the stored step 2 sort.

3.2 The Step 3 Sort

For $M = 16$ dimensions, the step 3 sort is fed a copy of the dot products of one subgroup of $s = 64$ orientations, which were found by the step 2 sort. There are two defining characteristics of the subgroups. They are the only regions where

1. Dot products less than 8 are found; and
2. Dot products greater than 8 are not found.

Dot products equal to 8 are found both inside and outside of the subgroups.

Note from Fig. 6 that the step 2 sort also finds the first sub-sub-group (dot product=4, minimum Hamming distance of 6 bit-signs) in each sub-group. The step 3 sort determines how many orientations, n, are in each sub-sub-group from the size of the first one. For $M = 16$ dimensions this is $n = 8$.

The "step 3 sort" starts by setting the diagonal of the $s \times s$ matrix equal to 4 (which is the minimum nonzero dot product between all even or all odd orientations). Then rows $i = n+1 \rightarrow s$ are sorted n times based on columns $1 \rightarrow 2n$ so that the columns with higher indices are considered first and columns with successively lower indices are used to break ties. After each row sort, the same reordering is applied to the columns. After the nth row sort, the order of columns $1 \rightarrow 2n$ are held fixed, and rows $2n+1 \rightarrow s$ are sorted n times in the same fashion. Then rows $3n+1 \rightarrow s$ are sorted, and so on. Note that only $s - 2n$ row sorts are needed since the first group of n orientations was handled by the step 2 sort and the last sub-sub-group is composed of the only n orientations remaining after the intermediate $s/n - 2$ sub-sub-groups have been found.

As with the step 2 sort, the step 3 sort can also be done in a more memory efficient way (which will be useful for $M \geq 32$ dimensions). This can be accomplished as follows.

Calculate the dot product of the first even/odd orientation with all other even/odd orientations found in the step 2 sort. Out of these, find the highest orientation index k for which the the dot product is less than or equal to 4. Calculate the dot products of orientations $i = 2 \rightarrow k-1$ and $j = i+1 \rightarrow k$. From these determine the value of n.

Discard previously calculated dot products. Calculate, and sort by, the dot products of orientations $n+2 \rightarrow s$ with orientation $n+1$. Find the highest orientation index k for which the dot product is less than or equal to 4. Calculate the dot product of orientations $i = n+2 \rightarrow k-1$ and $j = i+1 \rightarrow k$. Consider the diagonal to be 4 instead of 0, and sort rows $n+1 \rightarrow k$ n times with column $2n$ being the most important and columns with successively lower indices breaking ties. As before, you apply the same reordering to columns $n+1 \rightarrow k$ after each row sort.

Discard previously calculated dot products. Calculate, and sort by, the dot products of orientations $2n+2 \rightarrow s$ with orientation $2n+1$. Find the highest orientation index k for which the dot product is less than or equal to 4. Calculate the dot product of orientations $i = 2n+2 \rightarrow k-1$ and $j = i+1 \rightarrow k$. Consider the diagonal to be 4 instead of 0, and sort rows $2n+1 \rightarrow k$ n times with column $3n$ being the most important and columns with successively lower indices breaking ties. As before, you apply the same reordering to columns $2n+1 \rightarrow k$ after each row sort. And so on.

4. APPLICATION TO BOSLHS AND RESULTS

Once the maximally spaced list of orientations has been found, extending the BOSLHS algorithm of Dalbey and Karystinos [18] to higher dimensions is trivial. For $M = 16$ dimensions, a randomly ordered, maximally spaced list of bins to fill with points can be generated as follows:

1. Randomly permute the order of the groups of orientations.
2. Within each group, randomly permute the order of the subgroups of orientations.
3. Within each subgroup, randomly permute the order of the sub-sub-groups.
4. Within each sub-sub-group, randomly permute the order of the orientations.
5. Within each orientation, randomly permute the order of octants.

The average centered and wrap around L_2 discrepancies of several sampling methods for $M = 16$ dimensions are plotted in Fig. 8. The tested methods include tensor product sampling, MCS, JS, LHS, BOSLHS, and the Sobol sequence. As in lower dimensions, BOSLHS was the random sampling method with the lowest discrepancy. The Sobol sequence has lower discrepancy. However, as was shown in Fig. 2, the Sobol sequence is highly regular, which can bias results.

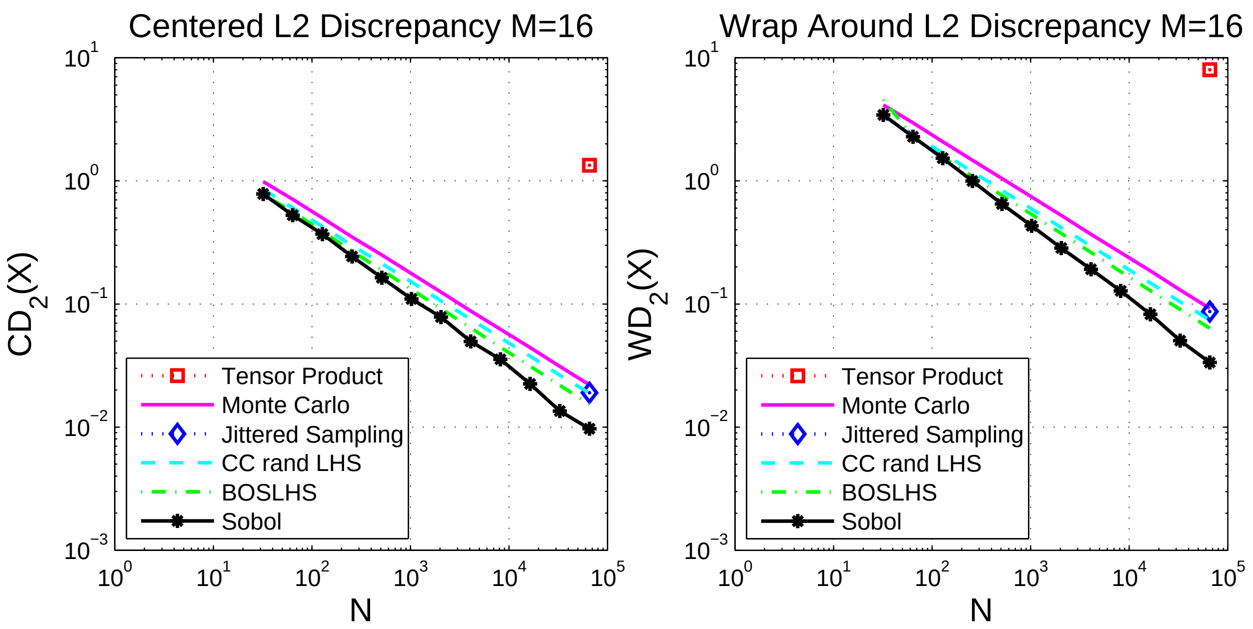

FIG. 8: Centered (left) and wrap around (right) L_2 Discrepancies as a function of the number of points N in $M = 16$ dimensions for tensor product sampling, MCS, JS, cell-centered Latin hypercube sampling with randomly paired dimensions, BOSLHS, and the Sobol sequence. Except for tensor product sampling and the Sobol sequence, which are both completely deterministic, the lines plot the average discrepancy for 40 randomly generated designs.

5. CONCLUSIONS

In the literature, space-filling Latin hypercube sample designs typically are generated by optimizing some criteria such as maximizing the minimum distance between points or minimizing discrepancy. However, such methods are time consuming and frequently produce designs that are highly regular, which can bias results.

A fast way to generate irregular space-filling Latin hypercube sample designs is to randomly distribute the sample points to a pre-selected set of well-spaced bins. Such designs are said to be "binning optimal" and are shown to be irregular. Specifically, Fourier analysis revealed regular patterns in the multi-dimensional spacing of points for the Sobol sequence but not for BOSLHS.

The ability to generate a list of well-spaced bins is a prerequisite for using this kind of sampling methodology. Simple patterns can be used to create a list of maximally spaced bins for $M = 2^r \leq 8$ dimensions and $N = 2^s \geq 2M$ points, where r and s are non-negative integers. The "patterns," if they can be called that, for maximally spaced octants in higher dimensions are not so simple.

In this paper, we presented a two-part procedure to generate a list of octants in maximally spaced order for $M \geq 16$ dimensions. The first part of the procedure uses group theory to generate a disjoint set of "orientations," where the term orientation refers to a rotated set of orthogonal axes. The octants/bins containing the $2M$ end points of an orientation are maximally spaced from each other. The second part is an algorithm to sort the orientations into maximally spaced order.

Conceptually, our procedure works for an arbitrarily large number of dimensions. However, memory requirements currently preclude even listing the $2^M/(2M)$ orientation leaders for $M \geq 32$ dimensions. In anticipation of overcoming this obstacle, we also outlined a variant of the sorting algorithm with a low memory requirement for use in higher dimensions.

The $M = 16$-dimensional BOSLHS designs generated using this list of maximally spaced octants have lower average centered and wrap around L_2 discrepancies than tensor product, Monte Carlo, jittered, and random cell centered Latin hypercube sample designs but higher discrepancy than the Sobol sequence. Discrepancy is a particularly useful sample design quality metric because of the Koksma-Hlawka-like inequality. It states that the error in a sample mean is bounded above by the product of the sample design's L_p discrepancy and the function's L_q variance, where $p^{-1} + q^{-1} = 1$.

ACKNOWLEDGMENTS

Sandia is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract No. DE-AC04-94AL85000.

REFERENCES

1. Hall, A., On an experimental determination of pi, *Messenger Math.*, 2:113–114, 1873.
2. Xiu, D. and Hesthaven, J. S., High-order collocation methods for differential equations with random inputs, *SIAM J. Sci. Comput.*, 27(3):1118–1139, 2005.
3. Chiu, K., Shirley, P., and Wang, C., Multi-jittered sampling, in *Graphics Gems Series*, vol. IV, pp. 370–374, Academic, San Diego, CA, 1994.
4. Park, J., Optimal Latin-hypercube designs for computer experiments, *J. Stat. Plan. Infer.*, 39(1):95–111, 1994.
5. Morris, M. and Mitchell, T., Exploratory designs for computational experiments, *J. Stat. Plan. Infer.*, 43(3):381–402, 1995.
6. Ye, K., Li, W., and Sudjianto, A., Algorithmic construction of optimal symmetric Latin hypercube designs, *J. Stat. Plan. Infer.*, 90(1):145–159, 2000.
7. Cioppa, T. M., Efficient nearly orthogonal and space-filling experimental designs for high-dimensional complex models, PhD thesis, Naval Postgraduate School, September 2002.
8. Bates, S., Sienz, J., and Langley, D., Formulation of the audze-eglais uniform Latin hypercube design of experiments, *Adv. Eng. Software*, 34(8):493–506, 2003.

9. Bates, S., Sienz, J., and Toropov, V., Formulation of the optimal Latin hypercube design of experiments using a permutation genetic algorithm, *Proc. of 45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Palm Springs, CA, April 19–22, Paper AIAA-2004-2011, 2004.
10. Husslage, B., van Dam, E., and Hertog, D., Nested maximin Latin hypercube designs in two dimensions, *CentER Discussion Paper No. 2005-79*, Tilburg University, Tilburg, The Netherlands, 2005.
11. Husslage, B., Rennen, G., van Dam, E., and Hertog, D., Space-filling Latin hypercube designs for computer experiments, *CentER Discussion Paper No. 2008-104*, Tilburg University, Tilburg, The Netherlands, 2006.
12. Cioppa, T. and Lucas, T., Efficient nearly orthogonal and space-filling Latin hypercubes, *Technometrics*, 49(1):45–55, 2007.
13. van Dam, E., Husslage, B., Hertog, D., and Melissen, H., Maximin Latin hypercube designs in two dimensions, *Oper. Res.*, 55(1):158–169, 2007.
14. Joseph, V. and Hung, Y., Orthogonal-maximin Latin hypercube designs, *Stat. Sin.*, 18(1):171–186, 2008.
15. van Dam, E., Rennen, G., and Husslage, B., Bounds for maximin Latin hypercube designs, *Oper. Res.*, 57(3):595–608, 2009.
16. Grosso, A., Jamali, A., and Locatelli, M., Finding maximin Latin hypercube designs by iterated local search heuristics, *Eur. J. Oper. Res.*, 197(2):541–547, 2009.
17. Lin, C., Bingham, D., Sitter, R., and Tang, B., A new and flexible method for constructing designs for computer experiments, *Ann. Stat.*, 38(3):1460–1477, 2010.
18. Dalbey, K. R. and Karystinos, G. N., Fast generation of space-filling Latin Hypercube Sample designs, In *Proc. of the 13th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, September, AIAA-2010-9085, 2010.
19. Lin, S. and Costello, D. J. J., *Error Control Coding: Fundamentals and Applications*, 2nd ed., Pearson Education, Upper Saddle River, NJ, 2004.