INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2010; **82**:135–156
Published online 12 October 2009 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nme.2750

An algorithm for fast optimal Latin hypercube design of experiments

Felipe A. C. Viana[1,*,†,‡], Gerhard Venter[2,§] and Vladimir Balabanov[3,¶]

[1]*Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL 32611, U.S.A.*
[2]*Stellenbosch University, Matieland, Stellenbosch 7602, South Africa*
[3]*The Boeing Company, Seattle, WA 98204, U.S.A.*

SUMMARY

This paper presents the translational propagation algorithm, a new method for obtaining optimal or near optimal Latin hypercube designs (LHDs) without using formal optimization. The procedure requires minimal computational effort with results virtually provided in real time. The algorithm exploits patterns of point locations for optimal LHDs based on the ϕ_p criterion (a variation of the maximum distance criterion). Small building blocks, consisting of one or more points each, are used to recreate these patterns by simple translation in the hyperspace. Monte Carlo simulations were used to evaluate the performance of the new algorithm for different design configurations where both the dimensionality and the point density were studied. The proposed algorithm was also compared against three formal optimization approaches (namely random search, genetic algorithm, and enhanced stochastic evolutionary algorithm). It was found that (i) the distribution of the ϕ_p values tends to lower values as the dimensionality is increased and (ii) the proposed translational propagation algorithm represents a computationally attractive strategy to obtain near optimum LHDs up to medium dimensions. Copyright © 2009 John Wiley & Sons, Ltd.

Received 14 January 2009; Revised 22 July 2009; Accepted 14 August 2009

KEY WORDS: design of computer experiments; experimental design; Latin hypercube sampling; translational propagation algorithm

1. INTRODUCTION

Design optimization usually requires a large number of potentially expensive simulations. Advancements in computational hardware and algorithms have not alleviated much of the resulting

*Correspondence to: Felipe A. C. Viana, Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL 32611, U.S.A.
†E-mail: fchegury@ufl.edu
‡Research Assistant.
§Professor.
¶Structural Analysis Engineer.

Contract/grant sponsor: Vanderplaats Research and Development, Inc.

computational crunch because of the ever increasing appetite for improved modeling of physical processes and more detailed optimization [1]. To reduce the computational cost, surrogate models, also known as meta-models, are often used in place of the actual simulation models [2–7]. Surrogate-based design optimization begins by identifying locations in the design space where simulations will be conducted. This process of identifying locations in the design space is known as design of experiments (DOE) [8, 9]. Response data (often via numerical simulations) are collected at these locations and one or more candidate surrogate models are fitted to the data [10–12]. Finally, one or more of the candidate models are selected for calculating responses (and to facilitate objective and constraint calculation during the optimization process) at points in the design space where the actual responses are not yet available.

It is well known among designers that the quality of fit, which often defines the performance of the surrogate model during optimization and design space exploration, strongly depends on the DOE (point location and density) [13, 14]. By quality of fit, the authors imply the discrepancy (in a general sense) between the actual response and the value predicted by the corresponding surrogate model. There exist many measures for the quality of fit, depending on the particular problem under consideration [14, 15].

A DOE with n_p points and n_v variables is usually written as an $n_p \times n_v$ matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n_p}]^{\mathrm{T}}$, where each row $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{\mathrm{in}_v}]$ represents a sample and each column represents a variable. Within the design and analysis of computer experiments, the Latin hypercube design (LHD) proposed by McKay *et al.* [16] and Iman and Conover [17] is very popular. The LHD presents advantages such as (i) the number of samples (points) is not fixed; (ii) orthogonality of the sampling points (a design is orthogonal if the inner product of any two columns is zero [3, 18]); and (iii) the sampling points do not depend on the surrogate model that will be constructed. An LHD with n_p points is constructed in such a way that each of the n_v variables is divided into n_p equal levels and that there is only one point (or sample) at each level. A random procedure is used to determine the point locations. Figure 1 shows two examples of LHDs with $n_v = 2$ and $n_p = 16$. As the LHD is constructed using a random procedure, there is nothing preventing a design that has poor space filling qualities, as the extreme case illustrated in Figure 1(a). A better choice is shown in Figure 1(b), where the points are more uniformly distributed in the domain.

The optimization of the space-filling qualities of the LHD is a challenging problem that has resulted in a number of research publications [19–28]. One interpretation of the space-filling property is to consider a n_v-dimensional sphere around each design point in the experimental design. The larger the radius of the smallest sphere that does not cross the boundary of the design space, the better the space-filling property of the design. Optimizing the point location in a LHD to

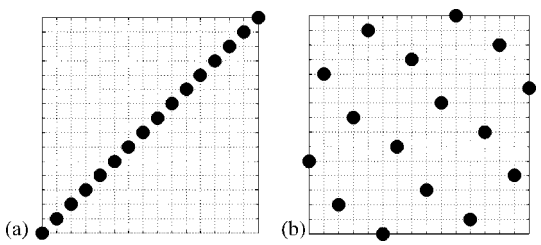

Figure 1. Examples of Latin hypercube designs: (a) ill-suited LHD with $n_v = 2$ and $n_p = 16$ and (b) reasonable LHD with $n_v = 2$ and $n_p = 16$.

DOI: 10.1002/nme

Table I. Approaches for constructing the optimal Latin hypercube design.

Researchers	Year	Algorithm	Objective functions
Audze and Eglājs [19]	1977	Coordinates Exchange Algorithm	Potential energy
Park [20]	1994	A 2-stage (exchange- and Newton-type) algorithm	Integrated mean-squared error and entropy criteria
Morris and Mitchell [21]	1995	Simulated annealing	ϕ_p criterion
Ye *et al.* [22]	2000	Columnwise–pairwise	ϕ_p and entropy criteria
Fang *et al.* [23]	2002	Threshold accepting algorithm	Centered L_2-discrepancy
Bates *et al.* [24]	2004	Genetic algorithm	Potential energy
Jin *et al.* [25]	2005	Enhanced stochastic evolutionary algorithm	ϕ_p criterion, entropy and L_2 discrepancy
Liefvendahl and Stocki [26]	2006	Columnwise-pairwise and genetic algorithms	Minimum distance and Audze–Eglājs functions
van Dam *et al.* [27]	2007	Branch-and-bound algorithm	1-norm and infinite norm distances
Grosso *et al.* [28]	2008	Iterated local search and simulated annealing algorithms	ϕ_p criterion

improve the uniformity of the point distribution, typically by maximizing the radius of the smallest sphere, results in an Optimal LHD. Such designs are usually obtained from time-consuming combinatorial optimization problems, with search space of the order of $(n_p!)^{n_v}$. For example, to optimize the location of 20 samples in two dimensions, the algorithm has to select the best design from more than 10^{36} possible designs. If the number of variables is increased to 3, the number of possible designs is more than 10^{55}. Researches have proposed various optimization algorithms and objective functions to solve this problem. Table I summarizes some strategies found in the literature. As for the computational time, Ye *et al.* [22] reported several hours on a Sun SPARC 20 workstation for generating an optimal Latin hypercube with 25 points for 4 variables. Jin *et al.* [25] reported minutes on a PC with a Pentium III 650 MHZ CPU for generating an optimal Latin hypercube with 100 points for 10 variables. Section 4 shows a comparison of different optimization methods for obtaining optimal LHDs with contemporary computing power.

Owing to its popularity, the ϕ_p criterion was selected as a performance measure in this paper. Minimizing ϕ_p leads to the maximization of the point-to-point distance in the design (see [21, 25] for details). Mathematically:

$$\phi_p = \left[\sum_{i=1}^{n_p-1} \sum_{j=i+1}^{n_p} d_{ij}^{-p} \right]^{1/p} \tag{1}$$

where p is a positive integer, n_p is the number of points in the design, and d_{ij} is the inter-point distance between all point pairs in the design. The general interpoint distance between any point pair $\mathbf{x}_i$ and $\mathbf{x}_j$ can be expressed as follows:

$$d_{ij} = d(\mathbf{x}_i, \mathbf{x}_j) = \left[\sum_{k=1}^{n_v} |x_{ik} - x_{jk}|^t \right]^{1/t}, \quad t = 1 \text{ or } 2 \tag{2}$$

In the present work, $p = 50$ and $t = 1$ are used following the suggestions of Jin *et al.* [25].

DOI: 10.1002/nme

Figure 1(b) shows what point locations one can expect to obtain if the ϕ_p criterion is used to generate an LHD with $n_v = 2$ and $n_p = 16$ (although other criteria may also lead to the same distribution of points).

In this work, the translational propagation algorithm is presented for obtaining optimum or near optimum LHDs. This algorithm requires minimal computational effort and does not use formal optimization. The aim is to solve the optimization problem in an approximate sense, that is, to obtain a good Latin hypercube quickly, rather than finding the best possible solution. The algorithm exploits simple translation of small building blocks (consisting of one or more points) in the hyperspace. The obtained LHD is referred to as TPLHD (a LHD obtained via the translational propagation algorithm). In general, the obtained TPLHD could be useful by itself as optimum or near optimum LHDs, or as an initial guess for numerical implementations based on some optimization algorithm.

The remainder of the paper is organized as follows. Section 2 describes the translational propagation algorithm for obtaining LHDs. Section 3 describes the numerical experiments used in this study. Section 4 presents the results and discussion. Finally, the paper is closed by recapitulating salient points and concluding remarks.

2. LATIN HYPERCUBE VIA TRANSLATIONAL PROPAGATION ALGORITHM

2.1. Basic algorithm

The proposed approach is based on the idea of constructing the optimal n_v-dimensional LHD from a fairly small n_v-dimensional seed design. The simple example of a 16×2 (i.e. 16 points in two dimensions) LHD is used to explain the methodology. Figure 2 shows some possible two-dimensional seed designs. While there is no limitation on the number of points for the seed design, the seed design can be as simple as just a single point ($1 \times n_v$ design). For this reason, the seed of Figure 2(a) is seed used in the 16×2 Latin hypercube example.

In order to construct a LHD of n_p points from a seed design of n_s points, the design space is first divided into a total of n_b blocks such that:

$$n_b = \frac{n_p}{n_s} \tag{3}$$

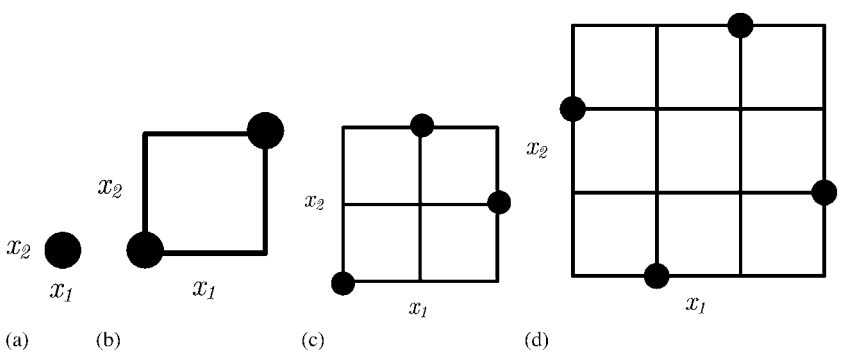

Figure 2. Example of seed designs for 2 variables: (a) 1×2 seed design; (b) 2×2 seed design; (c) 3×2 seed design; and (d) 4×2 seed design.

DOI: 10.1002/nme

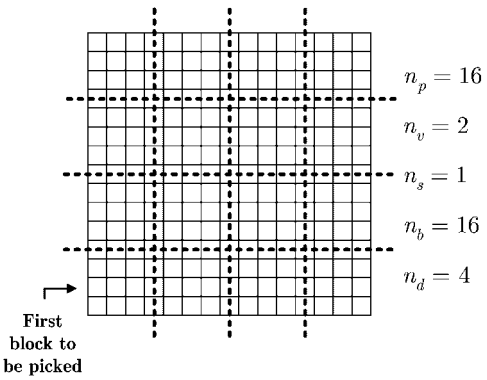

Figure 3. 16×2 Latin hypercube mesh divided into blocks (4 divisions in each dimension results in 16 blocks). The left-lower block is the first one to be picked in the algorithm.

Equation (3) means that each dimension is partitioned into the same number of divisions, n_d, calculated by:

$$n_d = (n_b)^{1/n_v} \tag{4}$$

In the example of the 16×2 LHD (i.e. $n_p = 16$ and $n_v = 2$), considering $n_s = 1$ (seed design selected from Figure 2(a)), one obtains $n_b = 16$ and $n_d = 4$ from Equations (3) and (4).

Next, each block is filled with the previously selected seed design. Figure 3 shows the division of the design space for the 16×2 LHD and which of the blocks is the first one to be picked in the algorithm. Figure 4 illustrates the process step by step. First, the seed design is properly scaled and placed at the origin of the first block, as shown in Figure 4(a). Next, the block with the seed design is iteratively shifted by n_p/n_d levels in one of the dimensions. Every time that the seed design is shifted, a new point is added to the experimental design. Figure 4(b) shows the first shift of the seed design (chosen to be in the horizontal direction). To preserve the Latin hypercube property of only a single point per level; Figure 4(b) shows that there also has to be a one-level shift in the vertical direction. To preserve the Latin hypercube properties, a displacement vector is built accounting for the shifting in the dimension of interest (horizontal direction in the example) as well a shift in all other dimensions (vertical direction in our example). The shifting process continues in one of the dimensions until all the divisions in this dimension are filled with the seed design as illustrated in Figure 4(a)–(d). In the next step, the current set of points (newly filled division) is used as a new seed design and the procedure of shifting the seed design is repeated in the next dimension. Figure 4(e)–(g) illustrates the shifting procedure in the vertical direction.

The biggest advantage of this approach is that there are no calculations to perform. All operations can be viewed as a simple translation of the seed designs in the n_v-dimensional hypercube. Although efficient for generating large designs, the algorithm proposed up to now fails to provide flexibility for the total number of points in the final LHD. The approach presented so far is limited in the sense that Equations (3) and (4) must hold. The next section describes a strategy to overcome this limitation and generate designs with *arbitrary number of points*.

DOI: 10.1002/nme

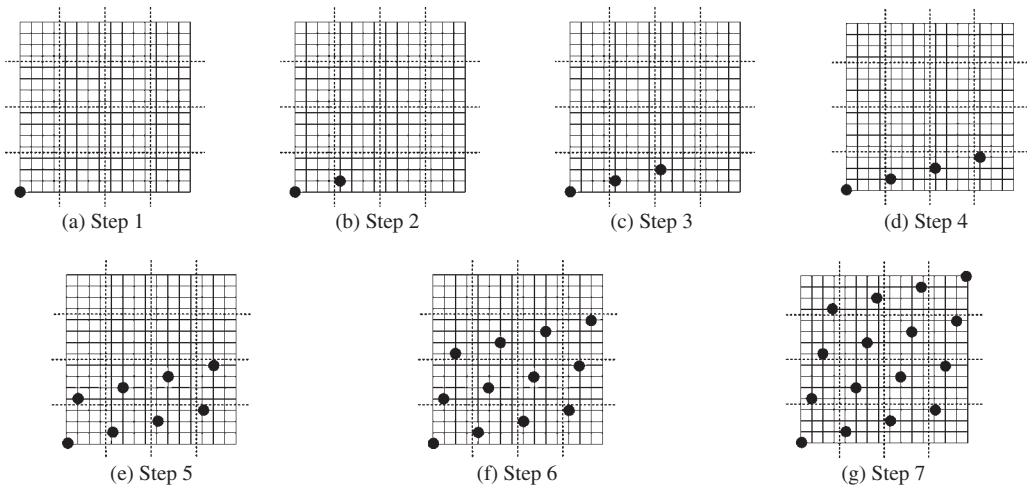

Figure 4. Process of creating the 16×2 enhanced Latin hypercube design. (a) illustrates the initial placement of the seed; (b)–(d) show the translation of the seed in the horizontal direction (which is accompanied by a one-level vertical displacement to preserve Latin hypercube properties); (d) also represents the newly created 'seed' that will be translated in the vertical direction; and (e)–(g) illustrate the translation in the vertical direction (which is accompanied by horizontal displacement of one level).

2.2. *Generating experimental designs of any size*

To generate an LHD with *any number of points*, the first step is to generate a TPLHD that has *at least* the required number of points using the algorithm described above. If this design contains the required number of points, the process is completed. Otherwise, an experimental design larger than the required is created and a *resizing* process is used to reduce the number of points to the desired one. The points are removed one by one from the initially created TPLHD by discarding the points that are the furthest from the center of the hypercube and reallocating remaining points to fill the whole design (preserving the Latin hypercube properties). In the proposed algorithm removing the points furthest from the center *does not* reduce the area of exploration. After removing the points, the final design is rescaled to cover the whole design space.

To illustrate the approach, consider the example where a 12×2 Latin hypercube is created using the one point seed of Figure 2(a) (i.e. $n_s = 1$). From Equations (3) and (4), $n_d = n_p^{1/n_v} \cong 3.46$ does not result in an integer number in this case. Rounding $n_d \cong 3.46$ up would give $n_d = 4$ and the next largest design that can be constructed is the 16×2, as illustrated in Figure 4. The resizing process begins with first calculating the distance between each of the 16 points and the center of the design space. To create a 12×2 design out of a 16×2 one, the four points furthest from the center have to be removed. In practice, this means the points of the original TPLHD have to be ranked according to the distance from the center of the design space. The suggested resizing algorithm computes these distances only one time, during the very first iteration. When a point is removed, the levels occupied by its projection along each of the dimensions have to be eliminated. This ensures that the Latin hypercube property that only a single point is found at any of the levels. Figure 5 illustrates the resizing process step by step. The number of points in the design actually

DOI: 10.1002/nme

shrinks, but the final design still represents samples over the same design space. Figure 5(a) shows that in the 16×2 design, the points in the left-bottom and right-top corners are equally far from the center. Owing to symmetry, it is not important which of the points will be removed first. The top-right point is removed first, Figure 5(a); and the bottom-left one is removed next, Figure 5(b). Figure 5(c) might be the best illustration of how the algorithm guarantees the Latin hypercube properties because it is the first time that an internal level is removed. Removing a point is as simple as eliminating it from the experimental design. However, as illustrated in Figure 6, this would leave empty levels (breaking the Latin hypercube requirements). The correct implementation of the resizing algorithm (Figure 5(c)) takes care of this limitation by also eliminating the levels that the point used to occupy. In Figure 5(c) this means that the three points on the right would move to the left (occupying the empty level that was in between the points). Next, the remaining points are scaled to cover the original design space. The scaling is as simple as the mapping of the remaining points to the design space of interest such that the lower and upper bounds are sampled by one of the points of the design (i.e. simple unidirectional scaling of all points). After each step, a new near optimum Latin hypercube is obtained with one point less. The process continues until the 12×2 design is achieved. Removing points/levels part of the algorithm reduces the number of points of the experimental design, while preserving the Latin hypercube requirements. On the other hand, the resizing part makes the experimental design to fit in the original design space again.

2.3. Seed designs

Using the described algorithm, one can create different instances of the TPLHD by changing the seed design, as illustrated by Figure 7 for the 16×2 TPLHD. As it is not known beforehand which seed design will lead to the best design in terms of the ϕ_p criterion, one would not know what seed design to use. However, this might be a positive feature of the algorithm. First, it is possible that the different designs are comparably good (when looking at the ϕ_p criterion). Second, due to the low cost of the algorithm (as we will show in the next sections), it pays to create instances of the Latin hypercube with different seeds and then pick the best one according to the ϕ_p criterion.

Because of the shifting process, the number of levels in a block is greater than the number of points in the seed design. This means that when creating design starting from seeds with more than one point ($n_s > 1$) it is necessary to reshape them to fit into one block of the TPLHD. Figure 8 illustrates this idea with the design of a 16×2 TPLHD starting from a 4×2 seed (i.e. $n_s = 4$). Figure 8(a) shows the initial 4×2 seed (that respects the Latin hypercube properties). It is necessary to insert as many empty levels between two consecutive projections of points as the number of division in a given directions, as shown in Figure 8(b). This way, at the end of the shifting process, there will be only one point per level (Figure 8(c)). The next section reviews the implementation of what was described so far.

2.4. Summary of the algorithm

The proposed algorithm is inspired by the optimization of the ϕ_p criterion penalizing designs with close points (due to $p = 50$ and $t = 1$ in Equation (1)). Good LHDs are expected to be obtained because the minimum distance is fixed at the level of seed design. This is particularly clear for large number of points in low dimensions. Figure 9 illustrates the translational propagation algorithm. The input parameters are the initial seed design $\mathbf{s}$, the number of points in the seed design, n_s,

DOI: 10.1002/nme

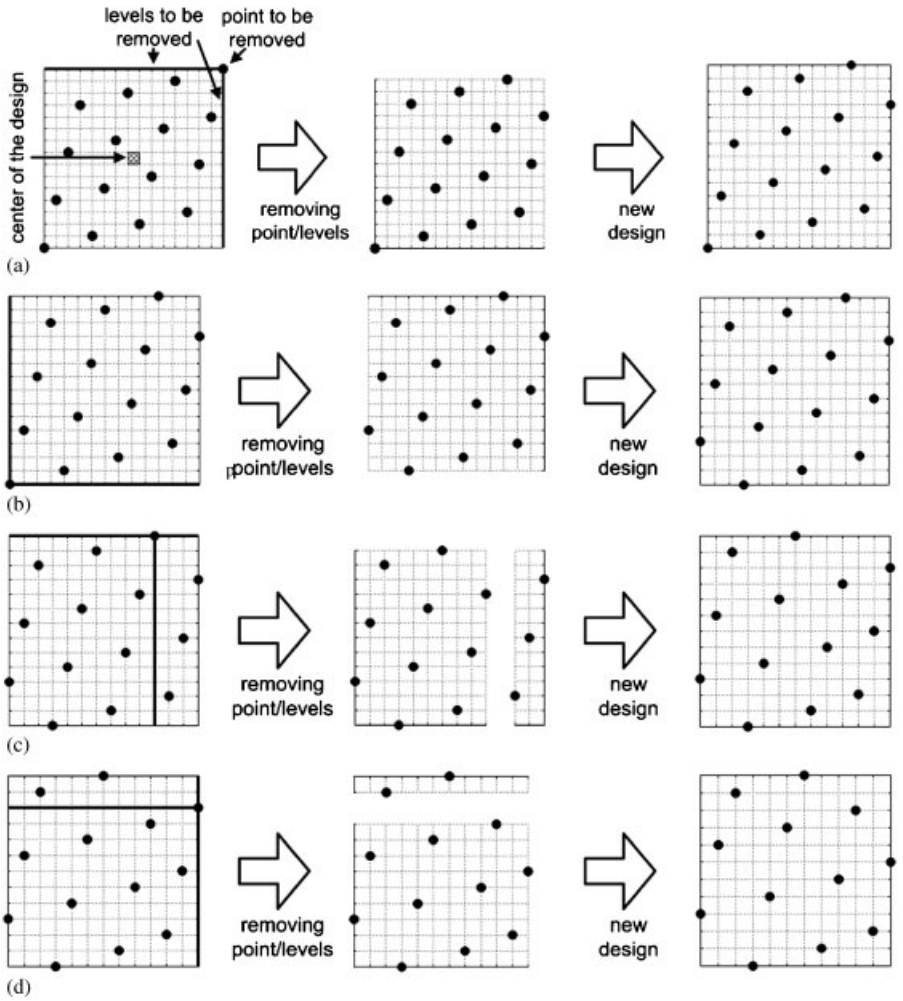

Figure 5. Process of creating a 12×2 enhanced Latin hypercube design from a 16×2 design.

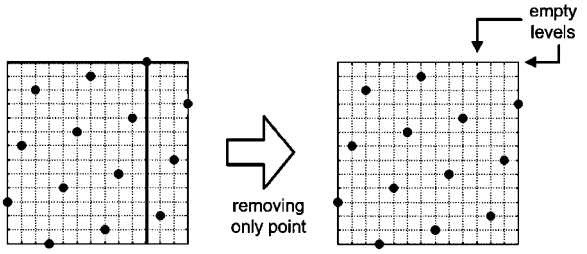

Figure 6. Wrong step of the resizing algorithm.

DOI: 10.1002/nme

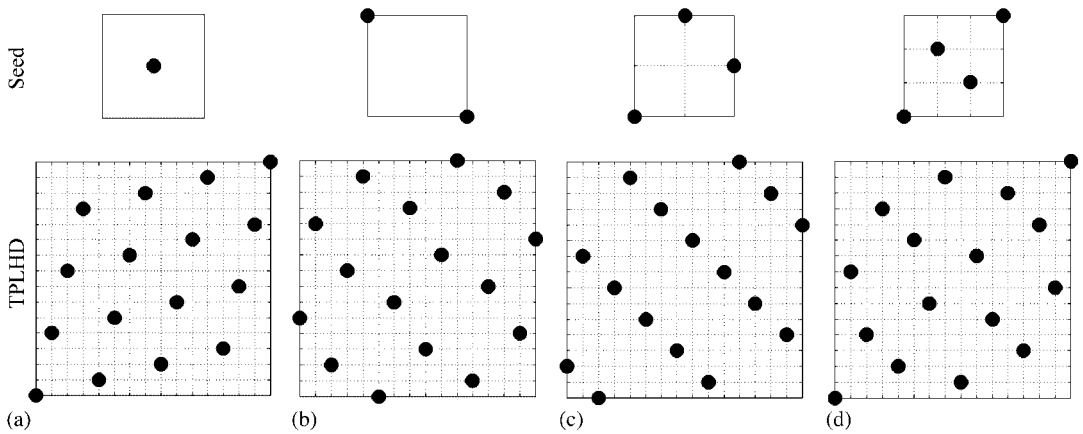

Figure 7. 16×2 TPLHD obtained with different seeds: (a) from 1-point seed; (b) from 2-points seed; (c) from 3-points seed; and (d) from 4-points seed.

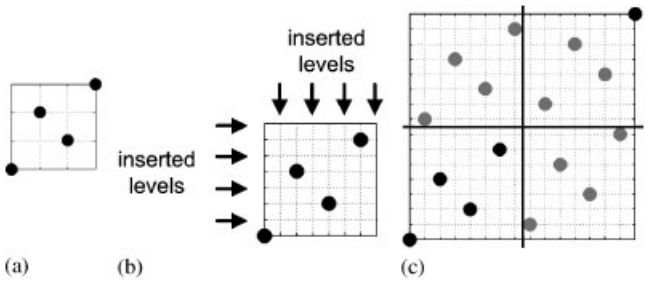

Figure 8. Illustration of the reshaping of the seed design: (a) original seed; (b) seed reshaped to fit in one block of the TPLHD; and (c) seed design filling different blocks of the TPLHD.

number of points of the desired LHD, n_p, and the number of variables, n_v. n_p^* and n_d^* are the number of points and number of divisions of the first TPLHD to be created. While n_p can assume any value, n_p^* is such that Equation (3) returns an integer number of blocks n_b. The first step is to check whether or not a bigger experimental design is needed. The number of divisions, n_d, and its rounded up value, n_d^*, are compared. If $n_d^* > n_d$, Equations (3) and (4) do not hold and the number of points in the TPLHD to be created, n_p^*, is greater than the desired, n_p. Next, the seed design is reshaped to fit in one block of the TPLHD (Section 2.3), and a TPLHD of n_p^* points is created (Section 2.1). Finally, if $n_p^* > n_p$, the TPLHD previously obtained is resized such that it will have n_p points (Section 2.2). $\mathbf{X}$ is the final TPLHD (with n_p points).

Appendix A gives the MATLAB implementation of the different building blocks of the translational propagation algorithm.

DOI: 10.1002/nme

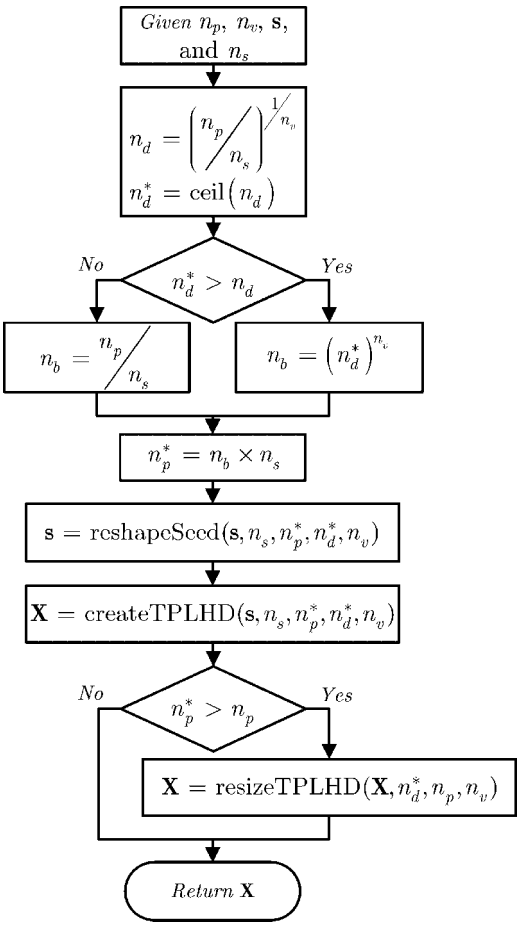

Figure 9. Flowchart of the translational propagation algorithm.

3. NUMERICAL EXPERIMENTS

To illustrate the effectiveness of the proposed TPLHD, a set of configurations that covers the typical application range of the LHD is considered. The experimental designs vary from only 2 to 12 variables as summarized in Table II. For each number of variables, three cases representing small, medium, and large number of design points were considered. The number of design points was calculated relative to the number of coefficients in a full polynomial model for the specified number of variables. The small designs have two times more points than the number of coefficients in a full quadratic polynomial model, while the large designs have 20 times more points than the number of coefficients in the same model. The medium designs have two times more points than the number of coefficients in a full cubic polynomial model.

The influence of the initial seed used to construct the TPLHD was studied first. For this study, different TPLHDs were generated using seeds with one to five points. The designs were then

DOI: 10.1002/nme

Table II. Latin hypercube design configurations considered.

No. of variables	No. of points		
	Small designs	Medium designs	Large designs
2	12	20	120
4	30	70	300
6	56	168	560
8	90	330	900
10	132	572	1320
12	182	910	1820

compared using the values of the ϕ_p criterion. Next, the efficiency of the proposed algorithm in approximating the optimal LHD was studied. For this study, an estimate of the range of the ϕ_p criterion for the LHDs proposed in Table II is created based on Monte Carlo simulation plus the results from 100 simulations of three different Latin Hypercube optimization techniques. The Monte Carlo simulation created 200 000 random LHDs for the configurations proposed in Table II (this is possible because the Latin hypercube algorithm allows for the creation of different designs based on a random number generator). The three different Latin Hypercube optimization techniques used in the study are:

1. The Enhanced Stochastic Evolutionary algorithm (ESEA) of Jin *et al.* [25]: set with maximum number of 20 iterations (but each simulation was allowed to run at most five iterations without improvement of the objective function).
2. The Genetic algorithm (GA) implementation of Bates *et al.* [24]: set with maximum number of 50 iterations (but each simulation was allowed to run at most 20 iterations without improvement of the objective function).
3. The native MATLAB function *lhsdesign* [29]: using the *maxmin* criterion (maximization of the minimum distance) and 200 iterations. With these settings, the MATLAB *lhsdesign* function selects the best LHD from 200 randomly created designs. The maxmin criterion is used to select the best design.

The settings for both GA and ESEA were based on a few trials for the design with 560 points and 6 variables. All simulations were conducted using an Intel Core 2 Quad CPU Q6600 at 2.40 GHz, with 3 GB of RAM running MATLAB 7.6 (R2008a) under Windows XP. The SURROGATES toolbox [30] was used to execute the TPLHD, ESEA, and GA algorithms under MATLAB [29].

Instead of showing only the ϕ_p criterion as defined in Equation (1), a normalized version, $\tilde{\phi}_p$, is also presented for easier comparison:

$$\tilde{\phi}_p = \frac{\phi_p - \min(\phi_p)}{\max(\phi_p) - \min(\phi_p)} \tag{5}$$

where $\max(\phi_p)$ and $\min(\phi_p)$ are the maximum and minimum values of ϕ_p found in the generated DOEs (including both the TPLHDs and the Monte Carlo simulations); which means that $0 \leqslant \tilde{\phi}_p \leqslant 1$.

Using the Monte Carlo simulations and the optimization results, it is possible to estimate the range of variation of ϕ_p values and thus make a judgment if a specific design represents a

DOI: 10.1002/nme

substantial improvement or not. The $\tilde{\phi}_p$ criterion makes the comparison of the distributions of different designs easier. Although it eliminates the sense of absolute magnitude (because of the normalization), it permits identifying if the distributions are toward the low or high values of the ϕ_p criterion.

4. RESULTS AND DISCUSSION

Table III shows the values of the ϕ_p criterion for TPLHDs obtained using different initial seed designs for design configurations of Table II. From these results, it can be concluded that no single seed size is always the best (even considering designs with the same number of variables). For example, the single point seed is the best for the 12×2 and 20×2 designs, while the two-point seed is the best for the 120×2 design. However, there might be cases where different seeds are equally competitive. As illustration, this is the case for the 30×4 design, where the four-point and five-point seeds produce designs with nearly the same ϕ_p criterion. These two observations together with the fact that no time-consuming optimization is required in the translational propagation algorithm suggests that it is possible to create several instances of the TPLHDs (based on different initial seeds) and select the design of choice based on the ϕ_p criterion. Table III also shows that an increase in dimensionality seems to favor the single point seed. The reason is that in higher dimensions a seed with multiple points acts like a cluster of points, hurting the ϕ_p criterion (this effect is less evident in lower dimensions).

Table IV shows detailed statistics regarding the ϕ_p criterion with data obtained during the Monte Carlo and optimization simulations (the 5th percentile is added to give information about the lower

Table III. ϕ_p criterion, defined in (1), ($p=50$, $t=1$) for TPLHD constructed from different seeds (the best TPLHD is shown in bold face).

		Seed size				
No. of variables	No. of points	1	2	3	4	5
2	12	**2.8**	2.9	3.8	3.7	3.8
2	20	**4**	4.8	4.9	5	**4**
2	120	11.0	**9.4**	12.6	13.4	13.8
4	30	1.9	1.8	1.8	**1.6**	1.6
4	70	2.7	5.0	2.9	**2.0**	2.1
4	300	7.2	13.6	6.2	**3.6**	4.0
6	56	**1.7**	2.1	1.8	3.7	1.7
6	168	3.1	6.0	**2.4**	2.9	**2.5**
6	560	**3.2**	11.7	5.3	7.9	3.6
8	90	**1.6**	3.4	2.6	5.9	1.9
8	330	3.7	4.6	3.1	3.0	**2.5**
8	900	4.7	16.4	4.6	2.7	**2.6**
10	132	**1.6**	3.7	3.9	6.6	3.5
10	572	**2.0**	8.8	4.0	3.1	2.9
10	1320	4.2	6.1	3.5	3.9	**3.1**
12	182	**1.7**	11.3	3.1	1.9	2.6
12	910	**2.0**	16.8	3.2	3.1	2.4
12	1820	**2.1**	17.8	3.7	3.4	2.4

DOI: 10.1002/nme

Table IV. TPLHD and statistics about the ϕ_p criterion, defined in (1), ($p=50, t=1$) for each studied configuration. Bold faces show when TPLHD offers the best design. TPLHD performs very well up to six dimensions.

				Percentiles					
No. of variables	No. of points	TPLHD	Min	5	25	50	75	Max	Range
2	12	**2.8**	2.3	3.7	5.5	5.6	5.6	5.7	3.4
2	20	**4.0**	3.4	6.4	9.5	9.6	9.7	10.0	6.6
2	120	**9.4**	9.4	40.2	59.5	60.3	60.8	62.3	52.9
4	30	**1.6**	1.5	2.3	2.9	3.2	3.7	7.4	5.9
4	70	**2.0**	2.0	3.8	4.9	5.8	6.9	17.3	15.3
4	300	**3.6**	3.4	8.6	11.1	13.0	15.7	74.8	71.3
6	56	**1.7**	1.0	1.5	1.8	2.0	2.3	7.9	6.9
6	168	**2.4**	1.3	2.4	2.8	3.2	3.6	23.9	22.6
6	560	**3.2**	1.8	3.7	4.4	4.9	5.7	32.9	31.1
8	90	1.6	0.9	1.1	1.2	1.3	1.5	5.9	5.0
8	330	2.5	1.4	1.6	1.8	2.0	2.2	9.7	8.3
8	900	2.6	1.8	2.1	2.4	2.7	3.0	16.3	14.5
10	132	1.6	0.7	0.8	0.9	1.0	1.1	6.6	5.8
10	572	2.0	1.0	1.2	1.3	1.4	1.5	8.8	7.8
10	1320	3.1	1.3	1.4	1.6	1.7	1.9	6.1	4.8
12	182	1.7	0.6	0.7	0.7	0.8	0.8	11.3	10.7
12	910	2.0	0.8	0.9	1.0	1.1	1.1	16.8	16.0
12	1820	2.1	0.9	1.0	1.1	1.2	1.3	17.8	16.9

values of ϕ_p). For up to six variables, TPLHD delivers exceptionally good designs (within 5% of the best designs for most of the cases). For 8–12 variables, TPLHD is not attractive anymore (with designs located within the last quartile of the ϕ_p criterion distribution). The reader will notice that for 10 variables the maximum value of ϕ_p drops when coming from 572 to 1320 points (contrarily from the trend of the other design configurations). This is because it is difficult to estimate this value for high-dimensional cases (even with total of more than 200 000 simulations). On the other hand, the percentile information establishes the trend of increasing ϕ_p with the number of points. As the algorithm was heuristically developed based on the visual observation of the patterns generated by the optimal LHD in two and three dimensions according to the ϕ_p criterion, it is of no surprise that the efficiency is harmed when the dimensionality increases. The range of the ϕ_p criterion varies with different design configurations and can also be very large (for example, the range for the 120×2 design is 52.9). Thus, only the normalized ϕ_p criterion, $\tilde{\phi}_p$, is used from this point on.

Figure 10 illustrates the box plots of the $\tilde{\phi}_p$ criterion for each of the studied configurations. A box plot shows the full data range on the y-axis. A box is defined by lines at the lower quartile (25%), median (50%), and upper quartile (75%) values. Lines extend from each end of the box for a distance of 1.5 times the interquartile range or until the data limit is reached. The data points falling outside this 1.5 times the interquartile range represent outliers that are shown using '+' symbols. Figure 10 shows that the distribution of $\tilde{\phi}_p$ tends to lower values as the dimensionality of the problem grows. It does not necessarily mean that the LHD fills the space better; after all, the sparsity of the points severely increases with the number of variables. However, this behavior

DOI: 10.1002/nme

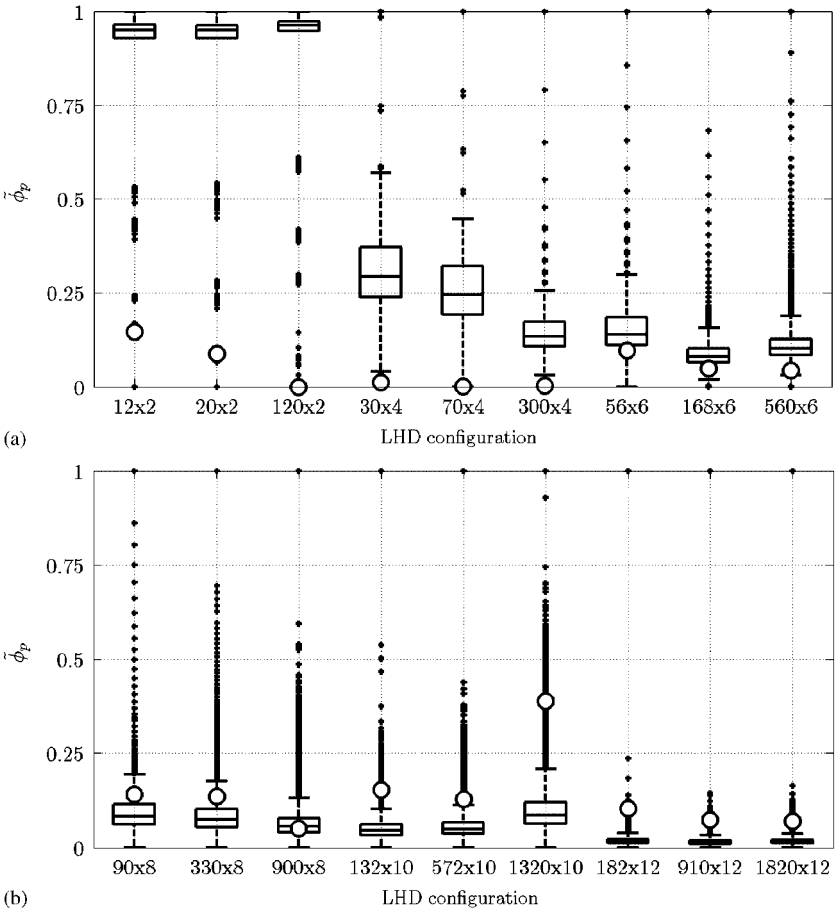

Figure 10. Boxplots of the $\tilde{\phi}_p$ criterion, defined in (5), $(p=50, t=1)$, where circles indicate the best TPLHD. TPLHD represents a good solution in low to medium dimensions. $\tilde{\phi}_p$ ranges from 0 to 1: (a) low to medium dimensions and (b) medium to high dimensions.

suggests that the probability of generating a design in the lower levels of the $\tilde{\phi}_p$ increases with the dimensionality. The following conclusions can thus be drawn from the numerical experimentation:

- In low dimensions: the optimum LHD is an outlier located in the lower region of the ϕ_p values; making it difficult to be obtained.
- In high dimensions: most of the LH designs are in the lower region of the $\tilde{\phi}_p$; making it difficult to distinguish the optimal LHD (obviously according to this criterion only) and other several good representations of the LHD.

Figure 10 makes clear that the TPLHD represents a better approximation of the optimum LHD in low to medium dimensions (up to six variables) and thus we will focus only on the designs with up to six variables.

DOI: 10.1002/nme

Table V. Performance comparison between TPLHD and the median out of 100 simulations of different optimization techniques. ESEA is the implementation of the Enhanced Stocastic Evolutionary algorithm of Jin *et al.* [25]. *lhsdesign* is a native MATLAB function. GA refers to the Genetic algorithm implementation of Bates *et al.* [24]. $\tilde{\phi}_p$, defined in (5), ranges from 0 to 1.

No. of variables		2			4			6		
No. of points		12	20	120	30	70	300	56	168	560
$\tilde{\phi}_p$	TPLHD	0.1	0.1	0	0	0	0	0.1	0	0
	ESEA	0.2	0.1	0.1	0	0	0	0	0	0
	GA	0.2	0.3	0.3	0.1	0.1	0	0	0	0
	lhsdesign	0.5	0.5	0.6	0.1	0.1	0.1	0.1	0.0	0.1
Time (s)	TPLHD	$\cong 0$	$\cong 0$	0.1	$\cong 0$	$\cong 0$	0.7	$\cong 0$	0.5	2
	ESEA	$\cong 0$	0.2	13	0.3	3	173	2	34	1096
	GA	0.4	0.9	24	4	17	275	19	143	1509
	lhsdesign	$\cong 0$	$\cong 0$	0.2	$\cong 0$	0.1	1.8	0.1	1.0	13

Table V allows a comparison between the TPLHD and the mean out of 100 simulations of three different optimization techniques: (i) the ESEA of Jin *et al.* [25]; (ii) the GA implementation of Bates *et al.* [24]; and (iii) the native MATLAB function *lhsdesign* [29]. TPLHD was obtained by generating five candidates from different seeds and then picking the best one according to the ϕ_p criterion (which means that Table V shows the time needed to generate all five candidates. Appendix B shows the discussion on the number of points that needs to be allocated in each of the cases; which directly impacts the computational cost). Results in Table V reinforce that TPLHD offers very good solutions; with most of the designs presenting $\tilde{\phi}_p = 0$. This means that TPLHD found the best result of the set of all simulations (including the Monte Carlo and all optimization ones). As for the computational cost, the TPLHD design is superior in most of the cases or in the worst-case matches the best time of the traditional optimization techniques. Clearly, the point density has a dramatic impact on the optimization techniques (especially ESEA and GA). For instance, for the ESEA with 6 variables, moving from 56 to 560 points makes the computational cost change from 2 s to a little more than 18 min. The *lhsdesign* MATLAB function tends to suffer less, as it can be seen as a Monte Carlo simulation with only 200 samples (200 is the number of iterations that *lhsdesign* was allowed to run). Still, in six dimensions, when moving from 56 to 560 points the computational cost of the *lhsdesign* increases more than 100 times.

Table V gives the impression that increasing the number of variables (no matter the point density) made the three optimization techniques to improve their own performance (in terms of the $\tilde{\phi}_p$ criterion). Considering the $\tilde{\phi}_p$ criterion of the designs with four and six variables, TPLHD and any of the optimization techniques would offer designs that are below 0.1. Then, the computational cost would leave only TPLHD and the *lhsdesign* as competing strategies. However, a closer look at the distributions of the $\tilde{\phi}_p$ criterion shows that TPLHD is the best choice. Figure 11 contrasts the box plot of the $\tilde{\phi}_p$ criterion with the values found by TPLHD and the 0.1 threshold of *lhsdesign*. Because the distributions tend to lower values, the 0.1 threshold is a bad value for the 300×4 design and is a marginal to undesired value in 6 dimensions. It is clear that TPLHD offers a better design (not mentioning a faster solution).

DOI: 10.1002/nme

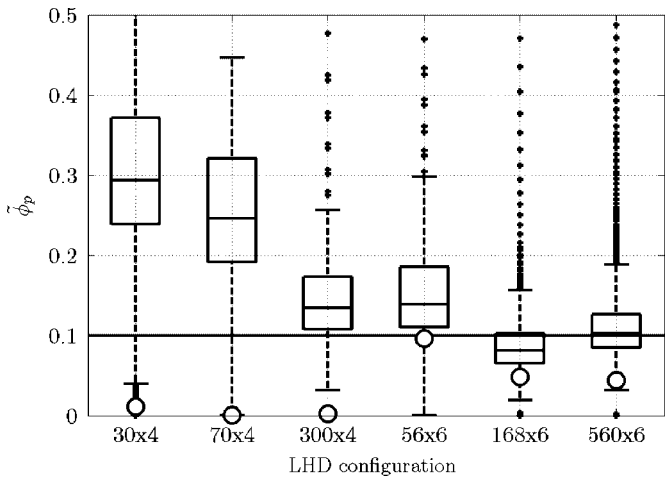

Figure 11. Boxplots of the $\tilde{\phi}_p$ criterion between 0 and 0.5 for the four and six-dimensional designs. $\tilde{\phi}_p$ is defined in (5), ($p=50$, $t=1$) and ranges from 0 to 1. Circles indicate the TPLHD.

One of the reasons for the poor performance of the TPLHD in high dimensions might be the existence of the directionality property of TPLHD. Because of the way the points are created in TPLHD, they tend to be stretched along one direction. Even the optimal LHD (obtained with ESEA or GA for example) from Figure 1(b) may be considered as having points placed in a preferred direction (close to 45°). Goel *et al.* [14] discussed that a single criterion for generating DOE may lead to large deteriorations in other criteria. In the proposed algorithm, the ϕ_p criterion is used and the directionality of the resulting LHDs (for both the TPLHD and the designs obtained with traditional optimization) is a clear loss. There has been work on the optimization of the space-filling properties of the LHD while preserving low correlation between variables. Cioppa and Lucas [31] presented an algorithm that improves the space-filling properties of LHD at the expense of inducing small correlations between the columns in the design matrix. However, authors warned that the approach is computationally prohibitive. Hernandez [32] presented an extensive study on a set of methodologies to create design matrices with little or no correlation (including saturated nearly orthogonal LHDs). Franco *et al.* [33] discussed a radar-shaped statistic for identifying the directionality property in a DOE (especially efficient in low dimension). However, there has been little research on how to employ this statistic for improving DOE and not just for identifying the directionality in an existing DOE, the directionality statistic was not employed to improve existing DOE in this paper. Nevertheless, the focus of this research is the translational propagation algorithm as a cheap alternative to traditional optimization of the LHDs based on the ϕ_p criterion.

Other, more general questions arise in higher dimensions: it is not clear what is the meaning of space-filling designs and the relative cost of any optimization strategy of experimental designs. The sparsity of the data may make it difficult to judge whether designs given by the optimization of the Latin hypercube are in fact space-filling designs. Even if they are, considering plots such as those in Figure 10, the computational cost of strategies such as TPLHD may end up being very close to random search (such as in *lhsdesign*). It may happen that in higher dimensions it is not worth investing much in time-consuming optimization (few iterations of random search

DOI: 10.1002/nme

may provide satisfactory results). In spite of that, authors intend to investigate the potential of the radar-shaped statistic in to improve the TPLHD properties.

5. CONCLUSIONS

A methodology for creating Latin hypercube designs (LHDs) via translational propagation algorithm (TPLHD) is proposed. The approach is based on the idea that a simple 'seed design' with a few points can be used as a building block to construct a near optimal LHD. The main advantage of the proposed methodology is that it requires virtually no computational time. The approach was tested on 18 different experimental design configurations with varying dimensionality and point density. Monte Carlo simulations were used to support the analysis of the algorithm's performance. For these cases it was found that:

- Based on the Monte Carlo simulations, the probability of randomly generating good designs in terms of the ϕ_p criterion (i.e. designs with low values of ϕ_p) increases with the dimensionality. This does not mean that the designs will fill the space better, it only means that designs tends to be equally competitive. This fact makes computationally cheap strategies such as random search attractive in high dimensions.
- It was found that for the low-dimensional cases (up to six variables) the TPLHD approximates the optimal LHD (using ϕ_p criterion) very well. In higher dimensions (8–12 variables), the TPLHD does not approximate the optimum solution well anymore.

As the time for generating the TPLHD is very small, in small to medium dimensions it is recommended to generate several TPLHDs (using seed designs with different number of points) and to pick the best one according to the ϕ_p criterion.

APPENDIX A: MATLAB IMPLEMENTATION OF THE TRANSLATIONAL PROPAGATION ALGORITHM

Tables AI to AIV give the MATLAB code of our implementation of the translational propagation algorithm. For simplicity, the levels of the Latin hypercube are represented by integer values. This means that each of the points of the created LHD can assume values from one to the number of points. The created Latin hypercube needs to be properly scaled to the design space of interest.

Table AI. MATLAB implementation of the translational propagation algorithm. The created design ranges from one to the number of points. The algorithm works with integer indexes for each point. `ceil(`x`)` is the MATLAB function that rounds the elements of x up to the nearest integers. `ones(`m`,`n`)` is the MATLAB function that creates an m-by-n matrix of ones. See Table AII for details about `reshapeSeed`. See Table AIII for details about `createTPLHD`. See Table AIV for details about `resizeTPLHD`. `vertcat`(A, B) is the MATLAB function that performs the vertical concatenation of matrices A and B.

```
1: function X = tplhsdesign(np, nv, seed, ns)
2: % inputs: np   - number of points of the desired Latin hypercube (LH)
3: %         nv   - number of variables in the LH
4: %         seed - initial seed design (points within 1 and ns)
5: %         ns   - number of points in the seed design
```


DOI: 10.1002/nme

Table AI. *Continued.*

```
% outputs: X   - Latin hypercube created using the translational
%          propagation algorithm

% define the size of the TPLHD to be created first
nd = ( np/ns)^( 1/nv ); % number of divisions, nd
ndStar = ceil( nd );
if (ndStar > nd)
    nb = ndStar^nv; % it is necessary to create a bigger TPLHD
else
    nb = np/ns; % it is NOT necessary to create a bigger TPLHD
end
npStar = nb*ns; % size of the TPLHD to be created first

% reshape seed to properly create the first design
seed = reshapeSeed(seed , ns, npStar, ndStar, nv);

% create TPLHD with npStar points
X = createTPLHD(seed, ns, npStar, ndStar, nv);

% resize TPLH if necessary
npStar > np;
if (npStar > np)
   X = resizeTPLHD(X, npStar, np, nv);
end
return
```

Table AII. MATLAB implementation of the `reshapeSeed` function. ones(m,n) is the MATLAB function that creates an m-by-n matrix of ones. `round`(x) is the MATLAB function that rounds the elements of X to the nearest integers.

```
function seed = reshapeSeed(seed , ns, npStar, ndStar, nv)
% inputs: seed     - initial seed design (points within 1 and ns)
%         ns       - number of points in the seed design
%         npStar   - number of points of the Latin hypercube (LH)
%         nd       - number of divisions of the LH
%         nv       - number of variables in the LH
% outputs: seed - seed design properly scaled

if ns == 1
    seed = ones(1, nv); % arbitrarily put at the origin
else
    uf = ns*ones(1, nv);
    ut = ( (npStar / ndStar) - ndStar*(nv - 1) + 1 )*ones(1, nv);
    rf = uf - 1;
    rt = ut - 1;
    a = rt./rf;
    b = ut - a.*uf;
```


DOI: 10.1002/nme

Table AII. *Continued.*

```
18:      for c1 = 1 : ns
19:          seed(c1,:) = a.*seed(c1,:) + b;
20:      end
21:      seed = round(seed); % to make sure that the numbers are integer
22:  end
23:
24:  return
```

Table AIII. MATLAB implementation of the `createTPLHD` function. ones(m, n) is the MATLAB function that creates an m-by-n matrix of ones. length(a) is the MATLAB function that returns the number of entries in the vector a. vertcat(A, B) is the MATLAB function that does the vertical concatenation of the matrices A and B.

```
1:   function X = createTPLHD(seed, ns, npStar, ndStar, nv)
2:   % inputs:  seed     - initial seed design (points within 1 and ns)
3:   %          ns       - number of points in the seed design
4:   %          npStar   - number of points of the Latin hypercube (LH)
5:   %          nd       - number of divisions of the LH
6:   %          nv       - number of variables in the LH
7:   % outputs: X  - Latin hypercube design created by the translational
8:   %               propagation algorithm
9:   % we warn that this function has to be properly translated to other
10:  % programming languages to avoid problems with memory allocation
11:
12:  X = seed;
13:  d = ones(1, nv); % just for memory allocation
14:  for c1 = 1 : nv % shifting one direction at a time
15:      seed = X;  % update seed with the latest points added
16:      d(1 : (c1 - 1))     = ndStar^(c1 - 2);
17:      d(c1)               = npStar/ndStar;
18:      d((c1 + 1) : end)   = ndStar^(c1 - 1);
19:      for c2 = 2  : ndStar % fill each of the divisions
20:          ns = length(seed(:,1)); % update seed size
21:          for c3 = 1 : ns
22:              seed(c3,:) = seed(c3,:) + d;
23:          end
24:          X = vertcat(X, seed);
25:      end
26:  end
27:  return
```

Table AI gives the main function, which has as input parameters the number of points, n_p, and variables, n_v, of the desired LHD, as well as the seed design, **s**, and its number of points, n_s. The points in the seed design initially assume values from one to the n_s. Lines 9–17 of the algorithm shown in Table AI illustrate how to take care of the size of the Latin hypercube to be created using the translational propagation algorithm. At first, a Latin hypercube observing Equations (3) and (4) is created; even if it means creating a design bigger than required. The seed design is then properly placed into the first block of to be filled by the algorithm (line 19). Line 43 is

DOI: 10.1002/nme

Table AIV. MATLAB implementation of the `resizeTPLHD` function. ones(m, n) is the MATLAB function that creates an m-by-n matrix of ones. zeros(m, n) is the MATLAB function that creates an m-by-n matrix of zeros. norm(x) is the MATLAB function that computes the vector norm ($\sqrt{\sum (x_i^2)}$). min (x) is the MATLAB function that finds the smallest element in x. sort(a) is the MATLAB function that sorts the vector a in the ascending order, returning the sorted and the index vectors. sortrows(X, COL) is the MATLAB function that sorts the matrix X based on the columns specified in the vector COL. isequal(A, B) is the MATLAB function that returns logical 1 (TRUE) if arrays A and B are the same size and contain the same values, and logical 0 (FALSE) otherwise.

```
1:  function X = resizeTPLHD(X, npStar, np, nv)
2:  % inputs: X      - initial Latin hypercube design
3:  %         npStar - number of points in the initial X
4:  %         np     - number of points in the final X
5:  %         nv     - number of variables
6:  % outputs: X     - final X, properly shrunk
7:
8:  center   = npStar*ones(1,nv)/2; % center of the design space
9:  % distance between each point of X and the center of the design space
10: distance = zeros(npStar, 1);
11: for c1 = 1 : npStar
12:     distance(c1) = norm( ( X(c1,:) - center) );
13: end
14: [dummy, idx] = sort(distance);
15: X = X( idx(1:np), : ); % resize X to np points
16:
17: % re-establish the LH conditions
18: Xmin = min(X);
19: for c1 = 1 : nv
20:     % place X in the origin
21:     X = sortrows(X, c1);
22:     X(:,c1) = X(:,c1) - Xmin(c1) + 1;
23:     % eliminate empty coordinates
24:     flag = 0;
25:     while flag == 0;
26:         mask = (X(:,c1) ~ = ([1:np]'));
27:         flag = isequal(mask,zeros(np,1));
28:         X(:,c1) = X(:,c1) - (X(:,c1) ~ = ([1:np]'));
29:     end
30: end
31: return
```

the call for the function that creates the initial LHD. If necessary, the design is then resized to the initially set configuration. Table AII gives the implementation of the reshaping of the seed design (see Section 2.3). Table AIII illustrates the body of the translational propagation algorithm. Lines 16–18 show how to create the displacement vector that is used to translate the seed in the hyperspace. Finally, Table AIV presents the implementation of the resizing process. Lines 8–15 show how we reduce the initially large Latin hypercube to the n_p initially set. Lines 18–30 present how to restore the Latin hypercube condition of one point per level to the remaining points.

DOI: 10.1002/nme

APPENDIX B: NUMBER OF POINTS ALLOCATED BEFORE RESIZING THE TPLHD

As discussed in Section 2, the number of points that needs to be allocated for a given configuration of the TPLHD is dictated by Equations (3) and (4). In the most general cases, a resizing process needs to be used. Thus, the total number of points initially generated dictates the computational cost of the TPLHD algorithm. Table BI shows the number of points allocated before the resizing process for each of the configurations studied in this paper. It can be observed that more the variables, the more the points are discarded. This confirms the fact that design strategies based on the domain subdivision (the translational propagation algorithm is an example) tend to have poor performance as dimensionality increases.

Table BI. Size of the TPLHD created before the resizing process for each seed size. The final best TPLHD is shown in bold face.

		Seed size				
No. of variables	No. of points	1	2	3	4	5
2	12	**16**	18	12	16	20
2	20	**25**	32	27	36	**20**
2	120	121	**128**	147	144	125
4	30	81	32	48	**64**	80
4	70	81	162	243	**324**	**80**
4	300	625	512	768	**324**	405
6	56	**64**	128	192	256	320
6	168	729	1458	**192**	256	320
6	560	**729**	1458	2187	2916	3645

ACKNOWLEDGEMENTS

The authors thank the members at Vanderplaats Research and Development, Inc. (VR&D) for the encouragement and support. This work was initiated when authors were employed by VR&D in various positions.

REFERENCES

1. Venkataraman S, Haftka RT. Structural optimization complexity: what has Moore's law done for us? *Structural and Multidisciplinary Optimization* 2004; **28**(6):375–387.
2. Sacks J, Welch WJ, Mitchell TJ, Wynn HP. Design and analysis of computer experiments. *Statistical Science* 1989; **4**(4):409–435.
3. Simpson TW, Peplinski JD, Koch PN, Allen JK. Meta-models for computer based engineering design: survey and recommendations. *Engineering with Computers* 2001; **17**(2):129–150.
4. Queipo NV, Haftka RT, Shyy W, Goel T, Vaidyanathan R, Tucker PK. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences* 2005; **41**:1–28.
5. Simpson TW, Toropov V, Balabanov V, Viana FAC. Design sand analysis of computer experiments in multidisciplinary design optimization: a review of how far we have come—or not. *Proceedings of the 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Victoria, Canada, 10–12 September 2008. AIAA 2008-5802.
6. Kleijnen JPC. *Design and Analysis of Simulation Experiments*. Springer: Berlin, 2007.
7. Forrester AIJ, Keane AJ. Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences* 2009; **45**(1–3):50–79.

DOI: 10.1002/nme

8. Montgomery DC. *Design and Analysis of Experiments*. Wiley: New York, 2004.
9. Simpson TW, Lin DKJ, Chen W. Sampling strategies for computer experiments: design and analysis. *International Journal of Reliability and Applications* 2001; **2**(3):209–240.
10. Viana FAC, Haftka RT, Steffen V. Multiple surrogates: how cross-validation errors can help us to obtain the best predictor. *Structural and Multidisciplinary Optimization*, DOI: 10.1007/s00158-008-0338-0.
11. Viana FAC, Haftka RT. Using multiple surrogates for metamodeling. *Seventh ASMO-UK/ISSMO International Conference on Engineering Design Optimization*, Bath, U.K., 7–8 July 2008.
12. Samad A, Kim K, Goel T, Haftka RT, Shyy W. Multiple surrogate modeling for axial compressor blade shape optimization. *Journal of Propulsion and Power* 2008; **24**(2):302–310.
13. Giunta AA, Wojtkiewicz SF, Eldred MS. Overview of modern design of experiments methods for computational simulations. *41st AIAA Aerospace Sciences Meeting and Exhibit*, Reno, NV, AIAA-2003-0649, 6–9 January 2003.
14. Goel T, Haftka RT, Shyy W, Watson LT. Pitfalls of using a single criterion for selecting experimental designs. *International Journal for Numerical Methods in Engineering* 2008; **75**(2):127–155.
15. Myers RH, Montgomery DC. *Response Surface Methodology. Process and Product Optimization Using Designed Experiments*. Wiley: New York, 1995.
16. McKay MD, Beckman RJ, Conover WJ. A comparison of three methods for selecting values of input variables from a computer code. *Technometrics* 1979; **21**:239–245.
17. Iman RL, Conover WJ. Small sample sensitivity analysis techniques for computer models, with an application to risk assessment. *Communications in Statistics*, *Part A. Theory and Methods* 1980; **17**:1749–1842.
18. Kleijnen JPC, Sanchez SM, Lucas TW, Cioppa TM. A user's guide to the brave new world of designing simulation experiments. *INFORMS Journal on Computing* 2005; **17**(3):263–289.
19. Audze P, Eglajs V. New approach for planning out of experiments. *Problems of Dynamics and Strengths* 1977; **35**:104–107. Riga, Zinatne Publishing House (in Russian).
20. Park JS. Optimal Latin-hypercube designs for computer experiments. *Journal of Statistical Planning and Inference* 1994; **39**:95–111.
21. Morris MD, Mitchell TJ. Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference* 1995; **43**:381–402.
22. Ye KQ, Li W, Sudjianto A. Algorithmic construction of optimal symmetric Latin hypercube designs. *Journal of Statistical Planning and Inference* 2000; **90**:145–159.
23. Fang KT, Ma CX, Winker P. Centered L2-discrepancy of random sampling and Latin hypercube design and construction of uniform designs. *Mathematics of Computation* 2002; **71**:275–296.
24. Bates SJ, Sienz J, Toropov VV. Formulation of the optimal Latin hypercube design of experiments using a permutation genetic algorithm. *Forty-Fifth AIAA/ASME/ASCE/AHS/ASC Structures*, *Structural Dynamics and Materials Conference*, Palm Springs, CA, 19–22 April 2004. AIAA-2004-2011.
25. Jin R, Chen W, Sudjianto A. An efficient algorithm for constructing optimal design of computer experiments. *Journal of Statistical Planning and Inference* 2005; **134**:268–287.
26. Liefvendahl M, Stocki R. A study on algorithms for optimization of Latin hypercubes. *Journal of Statistical Planning and Inference* 2006; **136**(9):3231–3247.
27. van Dam E, Husslage B, den Hertog D, Melissen H. Maximin Latin hypercube designs in two dimensions. *Operations Research* 2007; **55**(1):158–169.
28. Grosso A, Jamali A, Locatelli M. Finding maximin Latin hypercube designs by iterated local search heuristics. *European Journal of Operational Research* 2009; **197**(2):541–547.
29. Mathworks contributors, MATLAB® The language of technical computing, Version 7.6.0.324 R2008a, The MathWorks Inc., 2008.
30. Viana FAC. SURROGATES ToolBox, http://fchegury.googlepages.com, 2009.
31. Cioppa TM, Lucas TW. Efficient nearly orthogonal and space-filling Latin hypercubes. *Technometrics* 2007; **49**(1):45–55.
32. Hernandez AS. Breaking barriers to design dimensions in nearly orthogonal Latin hypercubes. *Ph.D. Thesis*, Naval Postgraduate School, Monterey, CA, U.S.A., 2008.
33. Franco J, Carraro L, Roustant O, Jourdan A. A radar-shaped statistic for testing and visualizing uniformity properties in computer experiments. *Proceedings of the Joint ENBIS-DEINDE 2007 Conference*: *Computer Experiments Versus Physical Experiments*, Turin, Italy, 11–13 April 2007.