European Journal of Operational Research 214 (2011) 683–696

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Stochastics and Statistics

Efficient space-filling and non-collapsing sequential design strategies for simulation-based modeling

K. Crombecq [a,b,*], E. Laermans [b], T. Dhaene [b]

[a] Department of Mathematics and Computer Science, University of Antwerp, Middelheimlaan 1, 2020 Antwerp, Belgium
[b] Department of Information Technology (INTEC), Ghent University – IBBT, Gaston Crommenlaan 8, 9050 Ghent, Belgium

ARTICLE INFO

Article history:
Received 15 November 2010
Accepted 19 May 2011
Available online 30 May 2011

Keywords:
Regression
Design of computer experiments
Experimental design
Sequential design
Space-filling

ABSTRACT

Simulated computer experiments have become a viable cost-effective alternative for controlled real-life experiments. However, the simulation of complex systems with multiple input and output parameters can be a very time-consuming process. Many of these high-fidelity simulators need minutes, hours or even days to perform one simulation. The goal of global surrogate modeling is to create an approximation model that mimics the original simulator, based on a limited number of expensive simulations, but can be evaluated much faster. The set of simulations performed to create this model is called the experimental design. Traditionally, one-shot designs such as the Latin hypercube and factorial design are used, and all simulations are performed before the first model is built. In order to reduce the number of simulations needed to achieve the desired accuracy, sequential design methods can be employed. These methods generate the samples for the experimental design one by one, without knowing the total number of samples in advance. In this paper, the authors perform an extensive study of new and state-of-the-art space-filling sequential design methods. It is shown that the new sequential methods proposed in this paper produce results comparable to the best one-shot experimental designs available right now.

1. Introduction

For many modern engineering problems, accurate high fidelity simulations are often used instead of controlled real-life experiments, in order to reduce the overall time, cost and/or risk. These simulations are used by the engineer to understand and interpret the behavior of the system under study and to identify interesting regions in the design space. They are also used to understand the relationships between the different input parameters and how they affect the outputs.

However, the simulation of one single instance of a complex system with multiple inputs (also called factors or variables) and outputs (also called responses) can be a very time-consuming process. For example, Ford Motor Company reported on a crash simulation for a full passenger car that takes 36 to 160 h to compute (Gorissen et al., 2007). Because of this long computational time, using this simulation directly is still impractical for engineers who want to explore, optimize or gain insight into the system.

We assume the system under study is a black box, with little or no additional information available about its inner working except for the output it generates. This means that, without running simulations, nothing is known about the behavior of the function, and no assumptions can be made about continuity or linearity or any other mathematical properties the system might have. A final assumption is that the simulator is deterministic, meaning that the same output is produced if the simulator is run twice with the same input values. This is not the same as saying that there is a complete absence of noise; indeed, noise may be introduced by the way the simulator models and discretizes the real world. It only implies that, even if there is noise in the simulation outputs, the noise will be identical for two simulation runs with the same inputs.

The goal of *global* surrogate modeling (or metamodelling) is to find an approximation function (also called a surrogate model) that mimics the original system's behavior, but can be evaluated much faster. This function is constructed by performing multiple simulations (called samples) at key points in the design space, analyzing the results, and selecting a model that approximates the samples and the overall system behavior quite well (Gorissen et al., 2010; Batmaz and Tunali, 2003). This is illustrated in Fig. 1.

Please note that *global* surrogate modeling differs from *local* surrogate modeling in the way the surrogate models are employed. In *local* surrogate modeling, local models are used to guide the optimization algorithm towards a global optimum (Regis, 2011). The local models are discarded afterwards. In *global* surrogate modeling, the goal is to create a model that approximates the

* Corresponding author at: Department of Mathematics and Computer Science, University of Antwerp, Middelheimlaan 1, 2020 Antwerp, Belgium. Tel.: +32 3 2653450; fax: +32 3 2653777.
E-mail addresses: Karel.Crombecq@ua.ac.be (K. Crombecq), Eric.Laermans@ugent.be (E. Laermans), Tom.Dhaene@ugent.be (T. Dhaene).

0377-2217/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2011.05.032

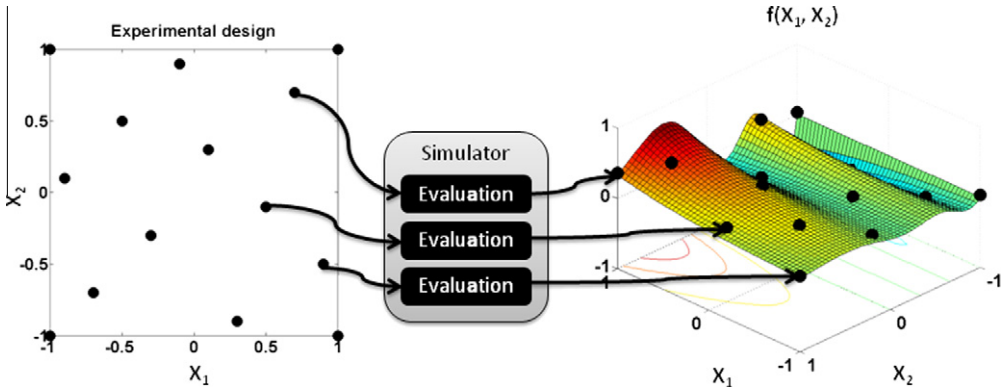


Fig. 1. A set of data points is evaluated by a black box simulator, which outputs a response for every data point. An approximation model (surrogate model) is fit to the data points, with the goal of minimizing the approximation error on the entire domain.

behavior of the simulator on the entire domain, so that the surrogate model can then be used as a full replacement for the original simulator, or can be used to explore the design space. Thus, the goal of global surrogate modeling is to overcome the long computational time of the simulator by providing a fast but accurate approximation, based on a one-time upfront modeling effort. In this paper, we are only concerned with global surrogate modeling.

Mathematically, the simulator can be defined as an unknown function $f : \mathbb{R}^d \rightarrow \mathbb{C}$, mapping a vector of d real inputs to a real or complex output. This function can be highly nonlinear and possibly even discontinuous. This unknown function has been sampled at a set of scattered data points $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\} \subset [-1,1]^d$, for which the function values $\{f(\mathbf{p}_1), f(\mathbf{p}_2), \ldots, f(\mathbf{p}_n)\}$ are known. In order to approximate the function f, a function $\tilde{f} : \mathbb{R}^d \rightarrow \mathbb{C}$ is chosen from the (possibly) infinite function set of candidate approximation functions F.

The quality of this approximation depends on both the choice and exploration of the function space F and the data points P. Ideally, the function f itself would be in the search space F, in which case it is possible to achieve an exact approximation. However, this is rarely the case, due to the complexity of the underlying system. In practice, the function $\tilde{f}$ is chosen according to a search strategy through the space F, in order to find the function that most closely resembles the original function, based on some error metric for the data points P (Busby et al., 2007; Jamshidi and Kirby, 2007).

It is clear that the choice of the data points P (called the experimental design) is of paramount importance to the success of the surrogate modeling task. Intuitively, the data points must be spread over the domain $\mathbb{R}^d$ in such a way as to convey a maximum amount of information about the behavior of f. This is a non-trivial task, since little or nothing is known about this function in advance.

It has been argued by some authors that the number of samples is much more important than the quality of the design (Liu, 2005). While it is obvious that the number of samples has an important effect on the quality of the model, it is clear that some experimental designs are better than others. Because every sample evaluation can potentially be very expensive, it is desirable to invest some time up front to determine the optimal location of a sample before submitting it for evaluation to the expensive simulator.

In this paper, we present a comparison and analysis of different space-filling sequential design methods. This study includes three novel methods developed by the authors and several other state-of-the-art methods from other authors. All these methods are compared against each other on a set of examples. The advantages and disadvantages of each method are discussed, and conclusions are drawn as to which method is preferable.

2. Sequential design

In traditional design of experiments (DoE), the experimental design P is chosen based only on information that is available before the first simulation, such as the existence of noise, the relevance of the input variables, the measurement precision and so on. This experimental design is then fed to the simulator, which evaluates all the selected data points. Finally, a surrogate model is built using this data. This is essentially a one-shot approach, as all the data points are chosen at once and the modeling algorithm proceeds from there, without evaluating any additional samples later on.

In the deterministic setting of computer experiments, well-known DoE techniques such as replication, randomization and blocking lose their relevance (Sacks et al., 1989). This leaves space-filling designs, which try to cover the domain as equally as possible, as the only interesting option. The advantages of the classic space-filling methods are that they can be easily implemented and provide a good (and guaranteed) coverage of the domain. Examples of popular space-filling designs are fractional designs (Simpson et al., 2001), Latin hypercubes (Grosso et al., 2009) and orthogonal arrays (Fang, 1980).

Sequential design (which is also known as adaptive sampling (Lehmensiek et al., 2002) or active learning (Sugiyama, 2006)) further improves on this approach by transforming the one-shot algorithm into an iterative process. Sequential design methods analyze data (models and samples) from previous iterations in order to select new samples in areas that are more difficult to approximate, resulting in a more efficient distribution of samples compared to traditional design of experiments.

Because the simulator is assumed to be a black box, it is infeasible in practice to predict how large the experimental design must be in order to achieve a given accuracy. Sequential design strategies solve this problem by selecting samples in an iterative manner, while at the same time updating (retraining) the model and assessing the quality of the model. If the model reaches the desired accuracy level, the sampling algorithm is halted, and no more samples are selected.

An essential consideration in sequential design is the trade-off between exploration and exploitation. Exploration is the act of exploring the domain in order to find key regions of the design space, such as discontinuities, steep slopes, optima, stable regions and so on, that have not been identified before. The goal is similar to that of a one-shot experimental design, in that exploration means filling up the domain as evenly as possible. Exploration does not involve the responses of the system, because the design space is defined by the inputs only.

The main advantage of exploration-based sequential designs over one-shot experimental designs is that the amount of samples evaluated depends on previous iterations of the algorithm. When one large experimental design is used, too many samples may have been evaluated to achieve the desired accuracy (oversampling) or too few samples may have been evaluated (undersampling), in which case one must completely restart the experiment or resolve to sequential methods to improve the initial design. Exploration-based sequential design methods will keep selecting new samples until the desired accuracy is found.

Exploitation is the other option. Instead of exploring the domain, data points are selected in regions which have already been identified as (potentially) interesting. For example, one might want to zoom in on optima, in order to make sure the surrogate model does not overshoot the optimum. Or one might also want to sample near possible discontinuities to verify that they are, in fact, discontinuous, and not just very steep slopes. Exploitation involves using the outputs of the previous function evaluations to guide the sampling process (Crombecq et al., 2009b).

The trade-off between exploration and exploitation is illustrated in Fig. 2. It is clear that without proper design space exploration, any sequential design strategy is bound to miss important regions in the response surface. Thus, every sequential design strategy must be space-filling to a certain degree. On top of this necessary foundation, exploitation-based methods can then zoom in on interesting regions to improve the generalization error in that region.

In this paper, we are only concerned with purely exploration-based (or space-filling) sequential design strategies. The goal is to see how exploration-based sequential design strategies stemming from different fields of research compare against each other, and how they hold up against a popular, proven space-filling experimental design called the Latin hypercube, which has some interesting mathematical properties.

3. Important criteria for experimental designs

From now on, we will consider the d-dimensional experimental design $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ containing n samples $\mathbf{p}_i = (p_i^1, p_i^2, \ldots, p_i^d)$ in the (hyper) cube $[-1, 1]^d$. With each experimental design, a construction method can be associated, which is used to construct this experimental design, given a particular dimensionality d and desired number of samples n. In order for this method to be a good space-filling sequential design strategy for computer experiments, it has to maximize three criteria.

3.1. Granularity

The first criterion is the granularity of the strategy. A fine-grained sequential design strategy can select a small number of points (preferably one) during each iteration of the algorithm. It should also generate reasonably space-filling designs, no matter after which iteration the algorithm is stopped. A coarse-grained sequential design strategy, on the other hand, selects new samples in large batches. The reason why a fine-grained method is prefered, is because it completely avoids over- or undersampling. When samples are only selected in large batches, too many samples may be evaluated at once, because only a few samples of the last batch were necessary to arrive at the desired prediction error. It is also possible that the modeler decides to stop the sampling before the desired prediction error is reached, because the next batch is too large and there is not enough computation time left to evaluate the entire batch.

Finally, the granularity of an algorithm also refers to the fact that the algorithm does not need to know the total number of samples that will be evaluated. Some methods, such as factorial designs and Latin hypercubes, require that the total number of samples be known in advance. In most real-life situations, this

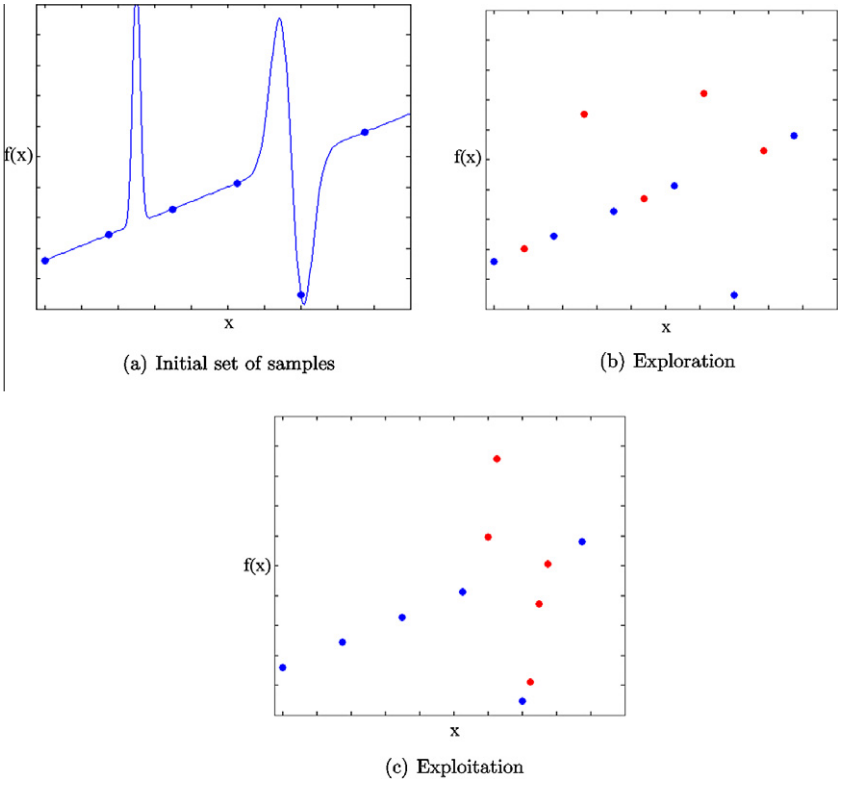


Fig. 2. This figure shows the trade-off between exploration and exploitation. In Fig. 2(a), a function and an initial set of samples are visualized. The function is unknown, and from looking at the samples, the function seems to behave linearly, except for one sample to the right. As illustrated in Fig. 2(b), exploration will explore the entire design space evenly, discovering new nonlinearities on the way. Exploitation, on the other hand, will focus on the area to the right because it seems to be the only nonlinear area, missing the additional nonlinearity to the left. This is depicted in Fig. 2(c).

information is unavailable, because the simulator is assumed a black box, and the complexity of the simulator, and the difficulty to model the problem, is not known up front. Therefore a good space-filling sequential design method should not make any assumptions about the maximum number of samples, and should work reasonably well no matter how many samples will be selected in the end.

3.2. Space-filling

Secondly, the generated design should be space-filling. Intuitively, a space-filling design is an experimental design X in which the points are spread out evenly over the design space. However, there are several ways to define this property mathematically. Over the course of the years, many different space-filling criteria have been proposed. The goal is to select the design P to maximize the criterion of choice. Depending on the criterion, the optimal design P will look differently. Table 1 gives an overview of the most popular ones, along with some references of people using the criterion. Some criteria might be used under different names in different publications; we use the most common name in this table and in further references in this article.

Note that most authors are concerned with finding an optimal design when the number of design points n is given and known in advance, and the entire design is generated at once instead of sequentially (i.e. worst possible granularity). In some cases (see Husslage, 2006 and Qian, 2009), the authors introduce some granularity in their methods, but they remain too coarse-grained for expensive computer experiments, in which each single sample evaluation may take hours and should be considered carefully.

Of these different criteria, the ϕ_p-criterion and the maximin criterion are the most widely used. The ϕ_p-criterion is an extension of the maximin criterion, introduced by Morris and Mitchell (1995) to differentiate between two designs which have the same maximin value. For large p, the ϕ_p criterion tends to rank designs in the same order as the basic maximin criterion, while for small p, the criterion behaves like the Audze-Eglais criterion. Additionally, if p is large enough, ϕ_p will differentiate between designs which have the same maximin value, but for which the second smallest distance between points is different. In this way, the ϕ_p-criterion is a family of criteria that encompasses both the maximin and Audze-Eglais criterion and everything in between.

However, the ϕ_p criterion has several disadvantages. The first problem is that it is numerically unstable in certain circumstances. When two points are very close to each other, and the power p is chosen large enough, ϕ_p will return infinity because of a floating point overflow. It is enough for one intersite distance to be rounded to zero, to result in a value of ϕ_p that is equal to infinity, no matter the quality of the rest of the design. The point at which this happens depends on both the design that is being rated and the number p, and the outcome is therefore difficult to predict in advance. This problem becomes an issue in sequential sampling, where the total number of samples (and therefore the intersite distance that is to be expected) is unknown up front.

Another problem with the ϕ_p criterion is that the value returned does not bear any geometrical meaning, and only provides a relative ranking between different designs. It is not intuitive to interpret the number and relate it to the density of the design. This also makes it difficult to combine the ϕ_p criterion with another one (for example: the projected distance criterion discussed in the next section). Additionally, the asymptotic nature of the ϕ_p criterion makes it very difficult to visualize the optimization surface, as the range of values is extremely small in most parts of the design space, and extremely large in small subparts.

Because of these issues, the ϕ_p criterion was not used in this study. The novel methods proposed in this paper combine multiple criteria to find an optimal solution for a multi-objective problem, and because of the lack of geometric meaning and the asymptotic nature of the surface, it is very difficult to combine the ϕ_p criterion with anything else. Instead, the maximin criterion, which does not suffer from any of the aforementioned issues, will be used to both generate and rank the designs. From now on, the maximin space-filling criterion will be refered to as the *intersite distance*, because it tries to maximize the smallest (Euclidean) distance any two sets of points (sites) in the design.

The problems with ϕ_p are not a major concern when this criterion is used to rank Latin hypercube designs, which already guarantee by construction a minimal distance between points. This explains why authors such as Viana et al. (2009) use the ϕ_p criterion without encountering any stability issues. They also do not combine the criterion with other criteria, because Latin hypercubes also guarantee good projective properties.

3.3. Good projective properties

Finally, a good space-filling design should also have good projective properties. This is also called the non-collapsing property by some authors (van Dam et al., 2007). An experimental design X has good projective properties if, for every point $\mathbf{p}_i$, each value p_i^j is strictly unique. This property also means that, when the experimental design is projected from d-dimensional space to $(d-1)$-

Table 1
Overview of different space-filling criteria.

Criterion	Formula
Manhattan (l_1 norm) van Dam et al. (2007), Ye et al. (2000)	$min_{\mathbf{p_i},\mathbf{p_j}\in P}\sum_{k=1}^{d} \mid p_i^k - p_j^k \mid$
Maximin (l_2 norm) van Dam et al. (2007), Joseph and Hung (2008) Morris and Mitchell (1995), Husslage (2006) Johnson et al. (1990), Ye et al. (2000)	$min_{\mathbf{p_i},\mathbf{p_j}\in P}\sqrt{\sum_{k=1}^{d} \mid p_i^k - p_j^k\mid^2}$
Audze–Eglais Audze and Eglais (1977)	$\sum_{\mathbf{p_i},\mathbf{p_j}\in P}\sqrt{\sum_{k=1}^{d} \mid p_i^k - p_j^k\mid^2}$
Centered L_2 discrepancy Hickernell (1998), Morris and Mitchell (1995), Jin et al. (2002) Fang et al. (2002), Fang and Lin (2003), Jin et al. (2005)	see Fang et al. (2002)
ϕ_p Morris and Mitchell (1995), Jin et al. (2002), Viana et al. (2009) Jin et al. (2005), Grosso et al. (2009)	$\left(\sum_{\mathbf{p_i},\mathbf{p_j}\in P}\sqrt{\sum_{k=1}^{d} \mid p_i^k - p_j^k\mid^2}^{-p}\right)^{1/p}$

dimensional space along one of the axes, no two points are ever projected onto each other.

The quality of a design in terms of its projective properties can be defined as the minimum *projected distance* of points from each other:

$$\|P\|_{-\infty} = min_{\mathbf{p_i},\mathbf{p_j}\in P} \min_{1\leqslant k\leqslant d} \left|p_i^k - p_j^k\right| = min_{\mathbf{p_i},\mathbf{p_j}\in P} \|\mathbf{p_i} - \mathbf{p_j}\|_{-\infty}, \quad (1)$$

where $\|x\|_{-\infty}$ is the minus infinity norm. This is a useful property if it is unknown up front if there are design parameters included in the experiment which have little or no effect on the response. If this is the case, two samples which differ only in this design parameter can be considered the same point, and evaluating this same point twice is a waste of computational time. Therefore, each sample should preferably have unique values for each design parameter. Ideally, when all the points are projected onto one of the axes the remaining design should be space-filling as well. Preferably, all the projected points should be equidistant. It is expected that an experimental design with optimal (equidistant after projection) non-collapsing points will not suffer a performance hit when one of the design parameters turns out to be irrelevant.

3.4. Orthogonality

Orthogonality is another desired property for an experimental design. A design P is called orthogonal with strength r if, for each subset of r inputs, each combination of different input values occurs the same number of times (Tang, 1993; Owen, 1992). This ensures that there is no correlation between the inputs in the design. Note that, according to this definition, only a small subset of possible designs can be orthogonal, namely those for which the input values are fixed at particular levels. The only designs included in this experiment that satisfy this condition are fractional factorial designs and Latin hypercube designs.

Additionally, for a given input dimension d and number of points n, an orthogonal design does not always exist. For the relatively small number of inputs and the (comparatively) large number of design points considered in this article, orthogonality cannot be satisfied. Even though Latin hypercubes can, by construction, never be completely orthogonal, they can be optimized such that submatrices of the hypercube are (Tang, 1993; Owen, 1992). Because orthogonality is irrelevant to most designs discussed in this article, this criterion will not be considered in this study.

4. Existing methods

In this section, we will discuss existing methods that will be used as benchmarks in this study. Both non-sequential methods, which have favorable properties in one or more of the criteria described in the previous section, as well as sequential methods from different fields of study will be investigated.

Each method will be given a name which will be used later in the discussion to refer to that particular strategy. Note that the design space is a hypercube with range $[-1,1]^d$, as opposed to the unit hypercube $[0,1]^d$, which is sometimes used in other studies. This has no effect on any of the algorithms, but may change some of the formulas.

4.1. Factorial designs

Factorial designs are the simplest form of space-filling designs (Montgomery, 2001). A full factorial design is a grid of m^d points. The full factorial is the best possible design in terms of the space-filling criterion; it maximizes the intersite distance for every number of m^d points. It is therefore expected that, if all the design parameters are equally important, a full factorial will produce the best results when used to train a model.

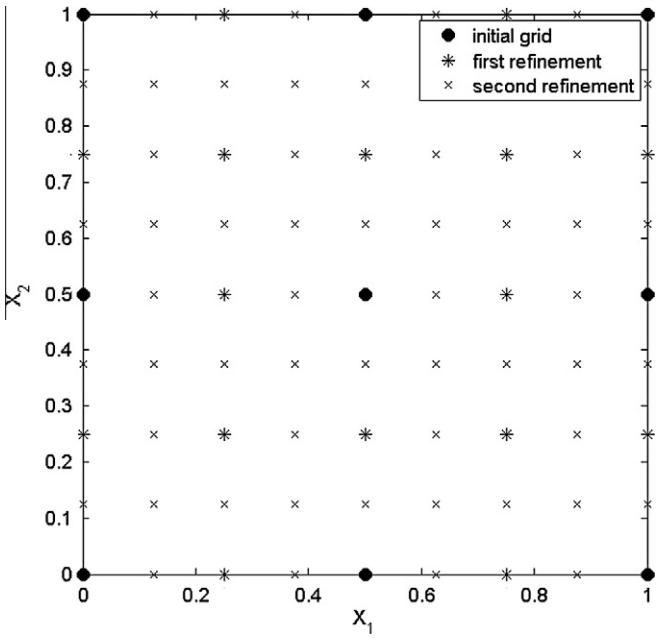


Fig. 3. A factorial refinement scheme.

However, it has several important disadvantages, which limit its practical use. Firstly, it is a very coarse-grained method: the full factorial can only be defined for the dth power of an integer m, which must be determined in advance. The only way to sequentialize a full factorial design is by evaluating the entire factorial design, and refine the grid in subsequent steps, as depicted in Fig. 3. This increases the size of the design by almost a factor 2^d at each iteration. Secondly, a factorial design has the worst possible projective properties: if one of the design parameters is unimportant, each unique point is evaluated m times. This is an unacceptable risk in a black-box setting.

To this end, several methods have been developed based on the factorial design, which tackle some of these issues. Fractional factorial designs remove some of the points from the grid, in order to limit the number of samples, making them feasible in high dimensional problems where a full factorial will take too much time to evaluate (Box et al., 2005). Latin hypercubes, which are discussed in the next section, can be considered a subclass of fractional factorial designs with additional requirements. In this study, the full factorial design with 12 levels will be considered for the 2-dimensional case (denoted as `factorial`), for a total of 144 points. The full factorial will be left out in higher dimensions because there is no full factorial with 144 points in these dimensions.

4.2. Latin hypercube

Latin hypercube designs (commonly denoted as LHDs (Viana et al., 2009)) are a very popular experimental design technique because of their well-understood mathematical properties, their ease of implementation and use and their speed. A Latin hypercube is constructed by dividing each dimension in the design space in m equally sized intervals, and placing exactly one point in each interval for each dimension. This construction method automatically results in an optimally non-collapsing experimental design. In addition, due to the stringent way in which the design space is divided, a Latin hypercube guarantees that each sample is at least $\frac{2}{m}\sqrt{2}$ away from the closest other sample in a $[-1,1]^d$ design space.

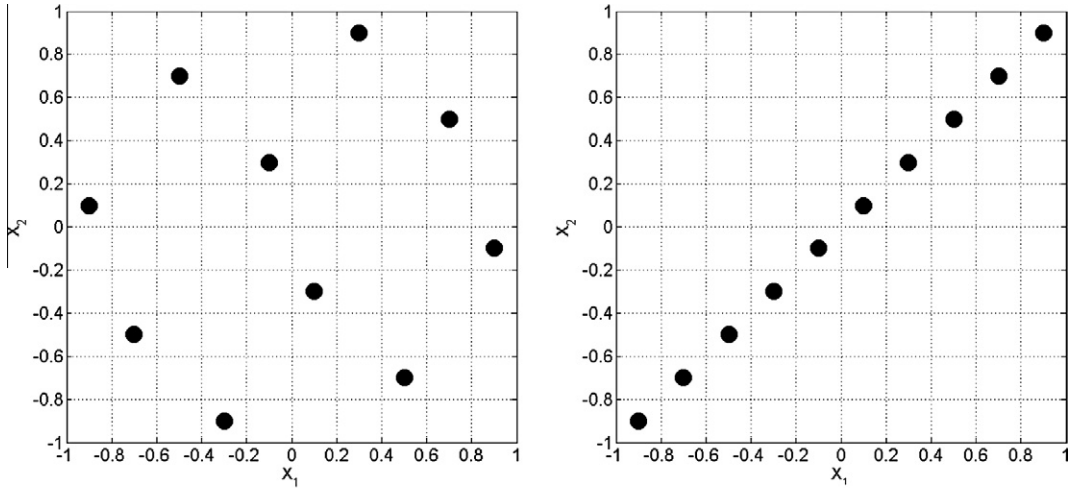


Fig. 4. Two different Latin hypercubes. While Fig. 4(a) has nice space-filling properties, Fig. 4(b) has only points in the diagonal and neglects two corners of the design space completely.

However, not every Latin hypercube has nice space-filling properties; this is illustrated in Fig. 4. Therefore, Latin hypercubes should not be used blindly and should be optimized according to a space-filling criterion. The optimization of Latin hypercubes is a very active research field, and many methods have been developed to reduce the search space. For a good overview of Latin hypercube optimization techniques, please refer to Viana et al. (2009).

It has been demonstrated by the authors that it is very difficult to generate a good space-filling Latin hypercube in reasonable time (Crombecq et al., 2009a). Even with state-of-the-art optimization algorithms, constructing a good space-filling Latin hypercube can take hours or even days. Husslage (2006) report that constructing a 100-point Latin hypercube in 3 dimensions took between 145 and 500 h on a P3–800MHz processor, depending on the algorithm used. For a larger number of points or higher dimensions, the computational time increases considerably.

To further verify the difficulty to generate a good Latin hypercube in a short timespan (e.g. 15 min), we included two different Latin hypercube generation methods in this study. The first method is an implementation of the optimization algorithm described in Joseph and Hung (2008) (denoted as `lhd-joseph`), which uses simulated annealing to optimize the intersite distance of the Latin hypercube. The second one uses the Matlab function `lhsdesign` from the Mathworks Statistics Toolbox to generate and optimize a Latin hypercybe (`lhd-matlab`).

Additionally, we also included the pre-optimized Latin hypercubes from Husslage (2006), van Dam et al. (2007), which can be downloaded from http://www.spacefillingdesigns.nl. All these Latin hypercubes were optimized for many hours to arrive at a semi-optimal or optimal solution. However, they are not available for every combination of dimensions and points. For our experiment, where we consider 144 points in 2 to 4 dimensions, a pre-optimized Latin hypercube is available on the website. We will refer to this Latin hypercube as `lhd-optimal`.

It is not straightforward to generate Latin hypercubes with a sequential design strategy. Firstly, the total number of samples must be determined in advance, in order to subdivide the design space into equally sized intervals. As mentioned before, this is an undesirable property, since there is little information available up front with which to make an educated guess on the required number of samples.

One way to sequentially generate Latin hypercubes is the idea of nested Latin hypercubes, in which one design is a subset of the other (Qian, 2009; Husslage, 2006). By using this method, the smallest subset can be evaluated first, after which the decision can be made whether the samples in the superset have to be evaluated as well. This can be extended to a number of so-called layers, in which each layer is a Latin hypercube in itself and is also a subset of the next layer.

This approach is not very suitable for our purpose, because it is not fine-grained enough. The technique proposed by Qian (2009) only allows for the size of each superset to be a multiple of the size of its subset, while Husslage (2006) only consider two layers of nested designs. Because one simulation is assumed to be expensive, a sequential design algorithm should preferably select samples one by one. Therefore, methods based on nested Latin hypercubes will not be included in this study. However, a similar, but new and more fine-grained method will be included in this study and is described in Section 5.1.

A second way to adapt Latin hypercubes to a sequential sampling process is to give up the requirement that each sample is placed in exact intervals, resulting in so-called quasi-Latin hypercubes. van Dam et al. (2007) define a class of quasi Latin hypercube designs in the design space $[0, n-1]^d$, based on a parameter $\alpha \in [0,1]$, which defines the minimum distance of samples from each other when projected onto one of the axes. For an α value of 0, this reduces to an unconstrained maximin design, while $\alpha = 1$ results in traditional Latin hypercubes. It was shown that the α value can be relatively close to 1 without reducing the space-filling qualities of the design much. Xiong et al. (2009) proposed a sequential design strategy in which the minimum projected distance of points from each other is reduced dynamically as more points are added to the design. A variation on this method will be included in this study, and will be described in Section 5.2.

4.3. Low-discrepancy sequences

Low-discrepancy sequences are sequences of points with the property that, for each n, the points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ have a low discrepancy. A set of points P has a low discrepancy if the number of points from the dataset falling into an arbitrary subset of the design space is close to proportional to a particular measure of size for this subset. Several definitions exist for the discrepancy, based on the shape of the subset, and the measure which is used. For more information on low-discrepancy sequences and different definitions for discrepancy, please refer to Niederreiter (1992), Jin

et al. (2005), Hickernell (1998). Low-discrepancy sequences are also called quasi-random sequences or quasi-Monte Carlo methods, because they can be used as a replacement for random uniform sampling.

Popular low discrepancy sequences have relatively good non-collapsing properties by construction. However, for small numbers of n, their space-filling properties are often subpar. Additionally, for some popular low-discrepancy sequences, such as the Hammersley sequence, the total number of points must be known in advance, because the points that are generated depend on the total number of points. So, for different values of n, completely different point sets are generated. Because these sequences are not suitable as a sequential design method, they will be ommited from this study.

Two of the most popular sequences that do not depend on the total number of samples are the `Halton` sequence and the `Sobol` sequence. The implementation of these sequences, that is available in the Matlab Statistics Toolbox, will be included in this study.

4.4. Remaining methods

The methods discussed and compared by Crombecq et al. (2009a) are also included in this study. These methods are very fine-grained, but only optimize towards intersite distance; they do not take into account the projected distance. The idea behind these methods will be discussed briefly in this section; for more information, please refer to Crombecq et al. (2009a).

The first method, `delaunay`, computes the delaunay triangulation of the samples, and selects a new sample in the center of gravity of the simplex with the largest volume. The second method, `voronoi`, estimates a Voronoi tessellation of the samples and selects a new sample in the largest Voronoi cell. Finally, `random` sampling is also included in the study, as a base case.

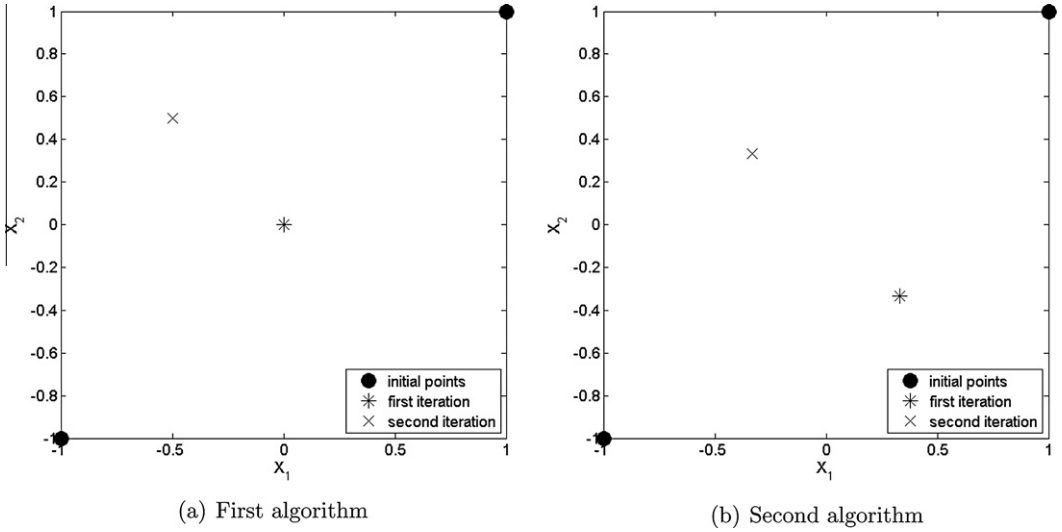
5. New space-filling sequential design methods

In this section, we propose a number of new space-filling sequential design methods that attempt to generate a design that scores well on both the space-filling criterion and the non-collapsing criterion, while being as fine-grained as possible (each method selects samples one by one). The goal of this study is to develop an algorithm that generates a design sequentially (one by one), with intersite and projected distance as close to the best non-sequential methods as possible. Additionally, this algorithm must run relatively quickly (at most 15 min for 144 points in 2D). This study was executed on an Intel Quadcore running at 1.86 GHz.

Because the new methods are sequential, they have to make do with a lot less information than their non-sequential counterparts (namely, the total number of samples is unknown in advance). Of course, this comes at a cost, and it is therefore expected that all the sequential methods will perform worse than pre-optimized Latin hypercube designs or factorial designs. However, if the drop in intersite distance and projected distance is small, these methods should be prefered over one-shot methods in a black-box environment, because they can avoid over- and undersampling, thus potentially saving lots of computational time. Additionally, some of the proposed methods will also work for very large n, for which optimizing a Latin hypercube is infeasible, and will also work for high dimensional problems.

At each iteration of a sequential design algorithm, the algorithm must determine the optimal location for the new sample point, based on the previously evaluated points. This new point must be located in such a way as to maximize the intersite and projected distance of the resulting design, which is composed of the original points and the new point. However, the new point must also ensure that future iterations of the algorithm can still produce good designs. Even if a point is optimally chosen for intersite and projected distance at one iteration, it might cause the algorithm to get stuck on a local optimum in subsequent iterations. This is illustrated in Fig. 5. This figure shows two 2D designs which were generated from the same set of two initial points: $(-1,-1)$ and $(1,1)$. The first algorithm places the third point in the origin, while the second algorithm places it at $(-0.3333,0.3333)$. After the third point, the first algorithm has produced the best design, both in intersite and projected distance. However, it is now stuck in a local optimum, as the best possible choice for the fourth point results in a design considerably worse than the one for the second algorithm.

This problem is further compounded by the difficult optimization surfaces produced by the intersite and projected distance. This is illustrated for 20 samples in 2D in Fig. 6. This figure shows the intersite and projected distance score of a 21-point design when the last point is moved over the entire 2D design space, while the previous 20 points are kept fixed. The intersite distance produces an optimization surface with a considerable number of local optima. But this does not even come close to the number of



(a) First algorithm (b) Second algorithm

Fig. 5. Two different sequential design algorithms generate a 4-point design starting from the same two initial points. The first algorithm gets stuck in a local optimum after the third point, while the second algorithm does not.

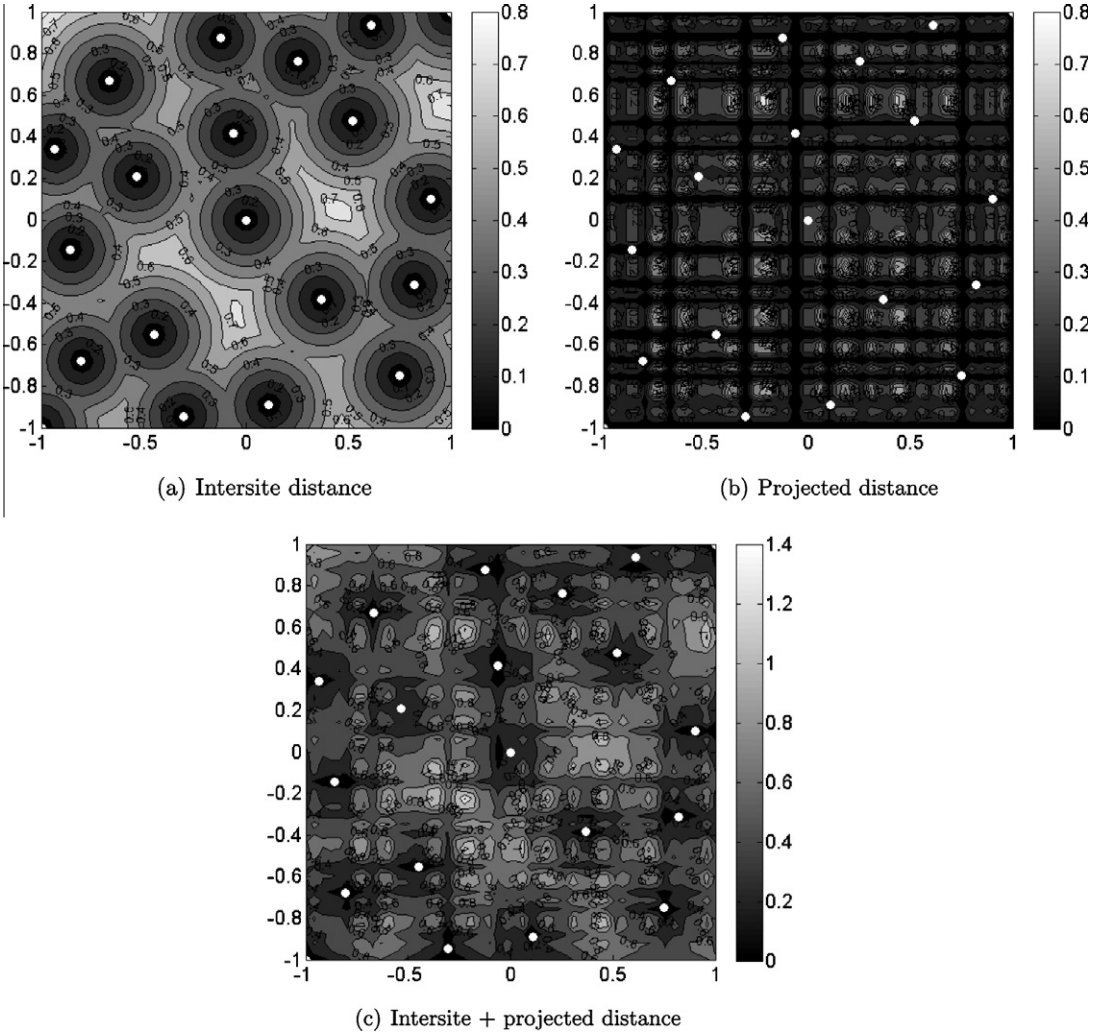


Fig. 6. The optimization surfaces for the intersite and projected distance criteria of a 20-point 2D design, as well as the sum of both criteria.

local optima that $\|P\|_{-\infty}$ has. In fact, $\|P\|_{-\infty}$ always has $(n+1)^d$ local optima, and only one of them is the global optimum. This optimization surface is so difficult, that it is practically impossible to optimize in an acceptable timeframe. When these two criteria are added, the resulting optimization surface is even more erratic.

Due to the extremely complex nature of this optimization surface, all the new methods proposed in this paper avoid working with this surface directly, by exploiting the structure of the projected distance surface, or by resorting to Monte Carlo methods instead of optimization. In the next sections, the new methods will be discussed in detail.

5.1. Sequential nested Latin hypercubes

A more fine-grained variant of the nested Latin hypercube method described in Section 4.2 was also included in this study. In order to sequentially generate Latin hypercube-like designs, one could refine the grid on which the points are chosen, similar to the idea of the factorial refinement scheme proposed in Fig. 3. Fig. 7 shows a refinement scheme for Latin hypercubes. Starting from an initial grid of m^d candidate points, m new samples are iteratively chosen on this grid. When a new sample is selected, all the other candidate points on the grid that have the same value for one of the design parameters are removed from the grid and will not be selected later. When m points have been selected, a new grid is created at the midpoints between the samples, and the process is repeated, thus (asymptotically) doubling the grid size at each iteration.

To determine which candidate point will be chosen next, the distance of each candidate point on the grid from all the previously selected points is computed. The candidate point that lies the farthest away is selected as the next sample, and all the other candidates that share one of the design parameter values with this sample are removed from the grid. Because the search space only contains the points on the grid instead of the entire design space, it is feasible to compute the distance for all the candidate points, without having to resort to optimization or Monte Carlo methods. This method is called `lhd-nested`.

This method results in an exact Latin hypercube when exactly $m + (m-1)(2^p - 1)$ samples have been selected, for $p > 0$. At these iterations, which depend solely on the initial grid size m, the $\|P\|_{-\infty}$ score is maximal. In the worst case, which is when $1 + m + (m-1)(2^p - 1)$ samples have been selected, the $\|P\|_{-\infty}$ score is almost half of the optimal score. When the total number of samples is known in advance, the number m can be tweaked such that the total number of samples is close to but not larger than $m + (m-1)(2^p - 1)$. However, in this study, this information is considered unknown, so m is fixed at 2.

5.2. Global Monte Carlo methods

A Monte Carlo method is a method that relies on repeated random sampling to compute the results. In the context of sequential design, Monte Carlo methods generate a large number of random

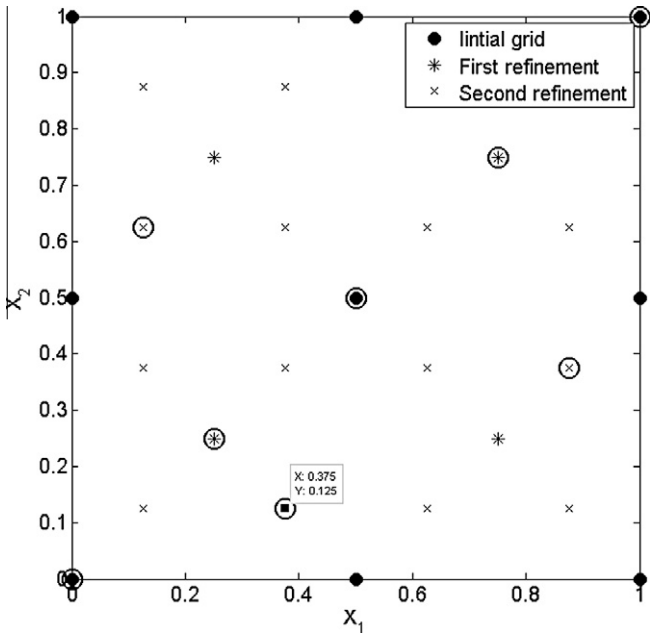


Fig. 7. A Latin hypercube refinement scheme, starting with $m = 3$. The points highlighted with a circle are the ones that were chosen by the sampling algorithm. Not that the design, composed of the encircled points, forms a Latin hypercube, and therefore has optimal projected distance.

candidate points in the design space, compute a criterion for all of these points, and select the point with the best (highest) score on the criterion as the next sample to be evaluated. This is repeated at each iteration. The number of random points is scaled with the number of samples that were previously evaluated: if the number of evaluated samples is n at one particular iteration, $100n$ random points are generated. The number 100 was chosen to keep the total computation time below 15 min.

Different criteria were tested in this study to rank the random points. The first criterion that was used is the aggregate of the intersite and projected distance, scaled to [0,1]. This criterion produces a score for a candidate design $P' = P \cup p$, which is composed of the previously evaluated samples P and a new candidate point p, according to the following formula:

$$\texttt{intersite}-\texttt{proj}-\texttt{base}(P') = \frac{\sqrt[d]{n+1}-1}{2}\min_{\mathbf{p_i},\mathbf{p_j}\in P'}\|\mathbf{p_i}-\mathbf{p_j}\|_2 + \frac{n+1}{2}\min_{\mathbf{p_i},\mathbf{p_j}\in P'}\|\mathbf{p_i}-\mathbf{p_j}\|_{-\infty}. \quad (2)$$

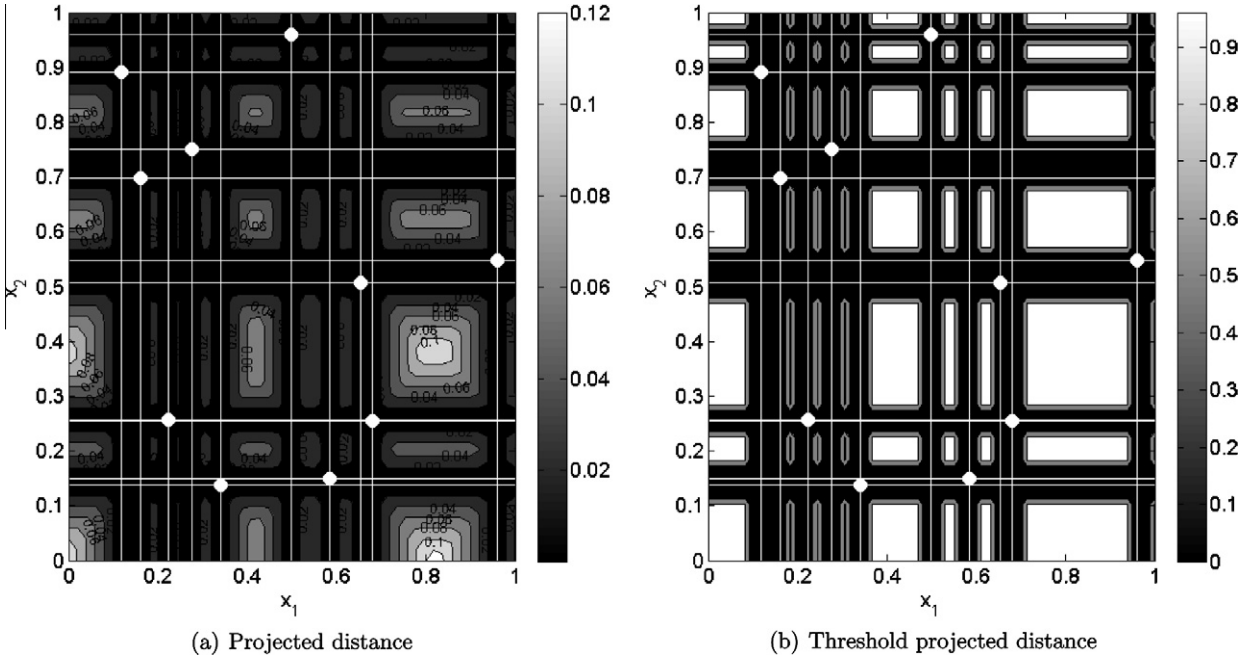
However, using this function as the objective is not yet ideal. Consider a design, for which two points already have an intersite distance of 0.1. Then all new candidates that lie further away from the other points than 0.1 result in the same intersite distance score, since the minimum intersite distance does not change. However, it is preferable to choose the point farthest away from the existing points. Therefore, instead of computing the distance of all points from each other, we just compute the distance of the new point from previous points, and optimize this function. The final objective function, which scores a new candidate point p based on the set of previously evaluated samples P, is defined as:

$$\texttt{intersite}-\texttt{proj}(P,\mathbf{p}) = \frac{\sqrt[d]{n+1}-1}{2}\min_{\mathbf{p_i}\in P}\|\mathbf{p_i}-\mathbf{p}\|_2 + \frac{n+1}{2}\min_{\mathbf{p_i}\in P}\|\mathbf{p_i}-\mathbf{p}\|_{-\infty}. \quad (3)$$

At each iteration, the point which maximizes the formula in Eq. (3) will be picked as the next point. This method will be refered to as `mc-intersite-proj`.

Note that the points are still ranked based on the complex surface shown in Fig. 6(a). An alternative is to consider the projected distance as a threshold function. The idea is to discard points that lie too close to other points in terms of projected distance. All the remaining points are then ranked solely on intersite distance. This is similar to the idea of Quasi Latin hypercube designs proposed in van Dam et al. (2007). The difference between the standard $\|P\|_{-\infty}$ criterion and the threshold projected distance criterion is shown in Fig. 8. In the case of the threshold criterion, only random points in the white areas are considered, and the best candidate in these areas based on the intersite distance is selected as the next sample.

The threshold, or minimum allowed projected distance, is defined as:



(a) Projected distance (b) Threshold projected distance

Fig. 8. The optimization surfaces for projected distance (Eq. (3)) and threshold projected distance criteria (Eq. (5)).

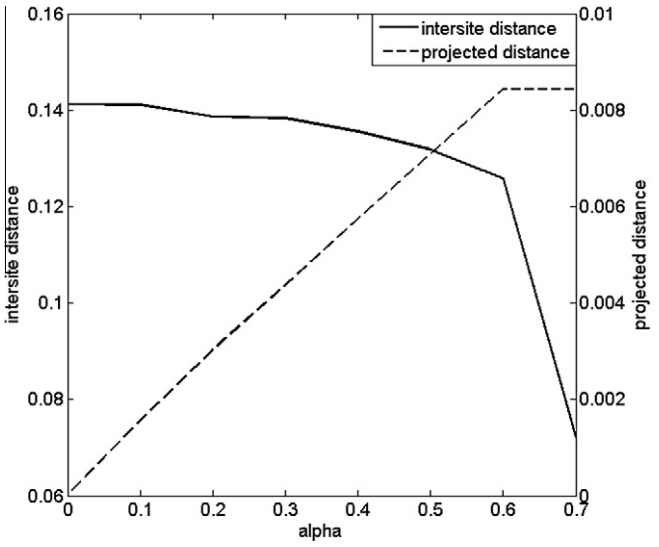


Fig. 9. The effect of the α parameter from the `mc-intersite-proj-th` algorithm on the intersite and projected distance. Lower values of α favour intersite distance, while higher values of α favour projected distance. For $\alpha = 0.5$, a good trade-off is found between intersite and projected distance.

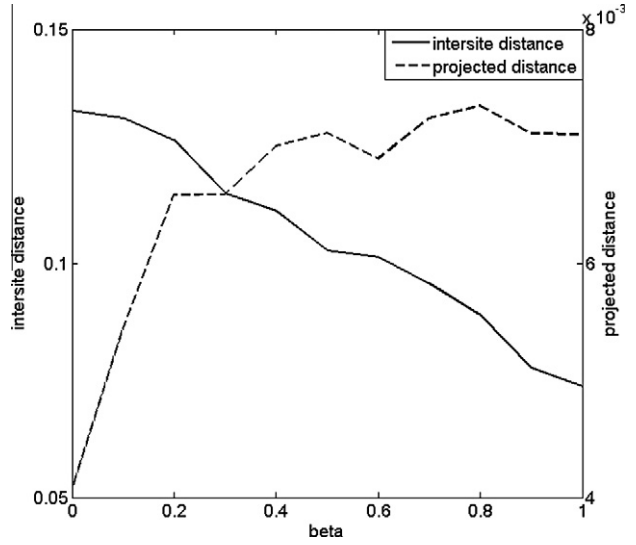


Fig. 10. The effect of the β parameter from the `optimizer-proj` algorithm on the intersite and projected distance. Lower values of β favour intersite distance, while higher values of β favour projected distance. For $\beta = 0.3$, a good trade-off is found between intersite and projected distance.

Table 2
The different space-filling design methods in terms of their granularity. It shows, for each method, if the method must know the total number of samples in advance, if the method is available for all number of samples and how many samples it selects at each iteration.

Method	# samples known	n restricted	Step size
`factorial`	yes	no	∞
`lhd-optimal`	yes	yes	∞
`lhd-nested`	no	yes	2^k
`voronoi`	no	yes	1
`delaunay`	no	yes	1
`random`	no	yes	1
`halton`, `sobol`	no	yes	1
`mc-intersite-proj`	no	yes	1
`mc-intersite-proj-th`	no	yes	1
`optimizer-intersite`	no	yes	1
`optimizer-proj`	no	yes	1

$$d_{min} = \frac{2\alpha}{n}, \tag{4}$$

where α is a tolerance parameter, which defines the importance of the projected distance. The objective function for the threshold version of Eq. (3) is defined as follows:

$$\texttt{intersite}-\texttt{proj}-\texttt{th}(P,\mathbf{p}) = \begin{cases} 0 & \text{if } \min\limits_{\mathbf{p_i}\in P} \|\mathbf{p_i}-\mathbf{p}\|_{-\infty} < d_{min}, \\ \min\limits_{\mathbf{p_i}} \in P \|\mathbf{p_i}-\mathbf{p}\|_2 & \text{if } \min\limits_{\mathbf{p_i}\in P} \|\mathbf{p_i}-\mathbf{p}\|_{-\infty} >= d_{min}. \end{cases} \tag{5}$$

If $\alpha = 0$, there are no constraints, and the projected distance is not taken into account at all. If $\alpha = 1$, only points that lie exactly on an optimal configuration are considered. In practice, this means that

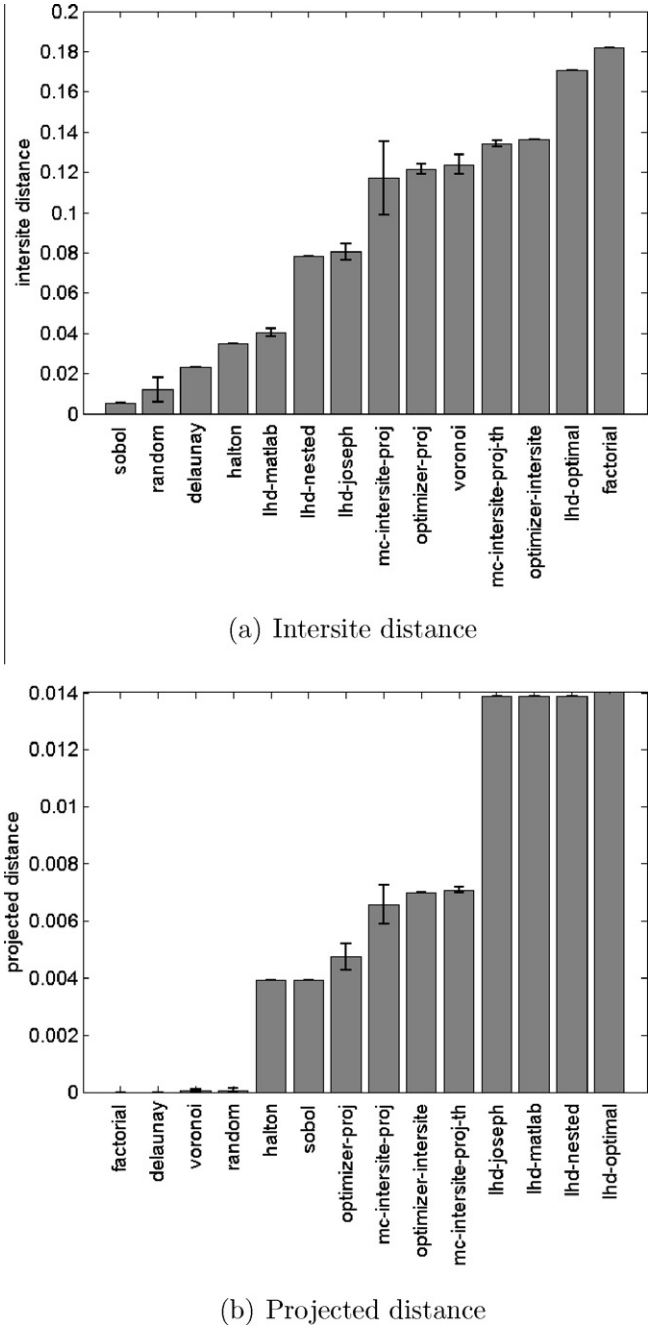


Fig. 11. The average intersite and projected distance score for each design method discussed in this paper, after generating a 144-point design in 2D.

all points are rejected, because the candidates are generated randomly. The trade-off between intersite and projected distance is illustrated in Fig. 9. For this experiment, $\alpha = 0.5$ was chosen because it results in a good trade-off between intersite and projected distance. The method using this objective function for ranking the candidate points will be refered to as `mc-intersite-proj-th`. This method can be further fine-tuned by adapting the algorithm to only generate random points in areas that fall outside of the threshold region, instead of eliminating the points after generation. This further improves the efficiency of this method.

5.3. Optimization-based methods

Even though global optimization methods do not seem to work well for this problem, local optimization methods can still deliver a considerable improvement when used after a Monte Carlo method. With this in mind, we propose two additional algorithms that perform a fast, constrained, local optimization after generating a large number of points, either based on a Monte Carlo method or based on the structure of the projected distance surface. We opted for the pattern search function from the Genetic Algorithm and Direct Search Toolbox of Matlab as the optimizer of choice, since it is a relatively fast but good optimizer that can get out of local optima quite easily.

5.3.1. Optimize projected distance locally

The first algorithm uses Monte Carlo to find the best points for the intersite distance, and then locally optimizes the best candidates for the projected distance, effectively giving up some intersite distance in exchange for better projected distance results. This method will be called `optimizer-proj`. Pseudo-code for this method can be found in Algorithm 1.

First, the algorithm selects a large amount of random points, and computes the intersite distance for all of these points. The 30 highest scoring points are selected as potential candidates, and the minimum distance from all the previously evaluated points is computed for these candidates. This distance is multiplied by a factor β which determines how much the optimizer may deviate from the selected candidate locations to improve the projective properties of the candidate. If β is set to 0, the algorithm selects points based solely on the intersite distance. If β is set to 1, the algorithm completely abandons the intersite distance and optimizes completely towards the projected distance. This trade-off is illustrated in Fig. 10. The β parameter effectively specifies how much space-fillingness the user is willing to give up for improved non-collapsingness. For this experiment, $\beta = 0.3$ was chosen because it provides a good trade-off between the two criteria.

Algorithm 1. The optimizer-proj algorithm

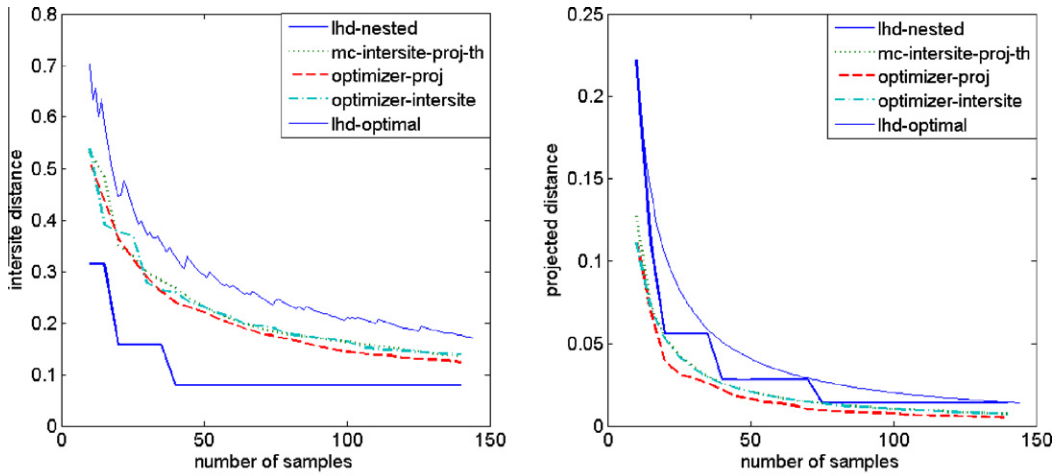
$P_{candidates} \leftarrow 100n$ random points
$P_{new} \leftarrow$ 30 best points using `intersite distance`
for all $\mathbf{p_{new}} \in P_{new}$ **do**
  $m(\mathbf{p_{new}}) \leftarrow min_{\mathbf{p}\in P} \|\mathbf{p_{new}} - \mathbf{p}\|_2$
  $d_{max} \leftarrow \frac{\beta m(\mathbf{p_{new}})}{2}$
  Optimize $\mathbf{p_{new}}$ towards $\|P \cup \mathbf{p_{new}}\|_{-\infty}$ on $[\mathbf{p_{new}} - d_{max}, \mathbf{p_{new}} + d_{max}]$
end for
Choose best $\mathbf{p_{new}}$ based on $\|P \cup \mathbf{p_{new}}\|_{-\infty}$

5.3.2. Optimize intersite distance locally

Even though the optimization surface of the $\|P\|_{-\infty}$ criterion is highly multimodal, it is also very structured, and the optima can easily be derived from the samples without having to use an optimization algorithm. Consider the intervals created by sorting all the values of the samples in one dimension, and substracting subsequent values. The point with the best projected distance score is then the point in the middle of the hypercube defined by the largest interval in each dimension. The second best point is the one created by replacing the one interval by the next largest, and so on. Once these points have been generated, a pattern search is performed in the 50 largest hypercubes, optimizing towards the intersite distance. The optimization surface is bound by a threshold parameter α, which works identical to the one defined in Section 5.2. Again, $\alpha = 0.5$ was picked because preliminary results have shown that this gives a good trade-off between intersite and projected distance. This method will be called `optimizer-intersite`.

6. Results

Each of the strategies mentioned in the previous sections will be used to generate 144 points in 2*D*,3*D* and 4*D*. Each method will be



(a) Intersite distance (b) Projected distance

Fig. 12. Respectively the intersite and projected distance as a function of the number of points selected so far. This graph shows the evolution over time as the algorithm selects more points, up to a maximum of 144 in 2D. For comparison, the intersite and projected distance of each pre-optimized Latin hypercube is also shown, even though it is not a sequential algorithm.

allowed to run at most 15 min to generate a design of 144 points on an Intel Quadcore running at 1.86 GHz.[1] This is acceptable, considering the fact that simulations are assumed to be expensive, and can take hours or days for one evaluation. In this context, 15 min to generate a good space-filling design is a good time investment. For each method in each dimension, the experiment will be run 30 times in order to get an estimate of the standard deviation on each method.

All the methods were compared on the three criteria discussed in this paper: granularity, space-filling and non-collapsing. The granularity of the methods is summarized in Table 2. Each new method proposed in this paper, except for `lhd-nested`, has the best possible granularity: the total number of samples does not have to be known in advance, they produce good designs whenever they are aborted, and they select samples one by one.

Fig. 11(a) contains the results for the intersite distance in 2D, after 144 points were generated. `factorial` is, of course, the best space-filling design. However, it is closely followed by `lhd-optimal`, which demonstrates that, if the total number of points in known in advance, it is possible to generate a design practically as space-filling as a factorial, but with optimal projected distance as well.

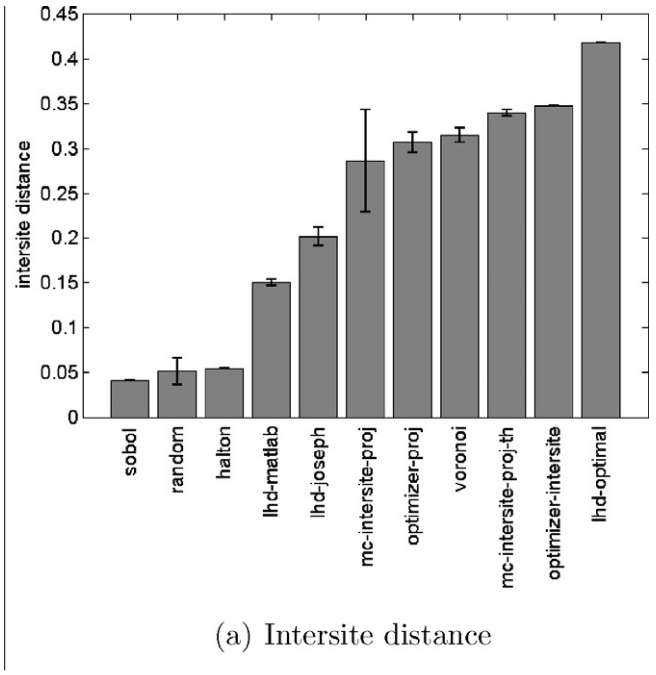
The next best methods are the four new methods proposed in this paper, as well as the `voronoi` algorithm. The best method turns out to be `optimizer-intersite`, which only performs 20% worse than the pre-optimized Latin hypercube, yet produced the design generated in a much smaller timespan, and with no knowledge at all of the total number of samples that were going to be needed. All the remaining methods perform much worse.

Note the big difference between the two sequential Monte Carlo strategies `mc-intersite-proj` and `mc-intersite-proj-th`. By replacing the projected distance by a threshold function, the quality of the design in terms of intersite distance improves considerably. The variance is also reduced, making the method much more stable. Also interesting to note is the rather poor performance of the Matlab Latin hypercube implementation, which was allowed to optimize for 15 min to allow for a fair comparison. This method fails at generating a good space-filling design, and should be avoided. The same can be said for the low-discrepancy sequences, which, even though they generate good space-filling designs for large numbers of points, perform bad for small sample sizes. Also noticeable is the bad performance of `lhd-nested`. This can be explained by the fact that, by selecting the optimal point from the Latin hypercube grid at one iteration, future iterations may get stuck in a local optimum, as described in Section 5. In this case, the last point selected before the grid is refined will be a very bad choice, resulting in a dramatic drop in quality of the design.

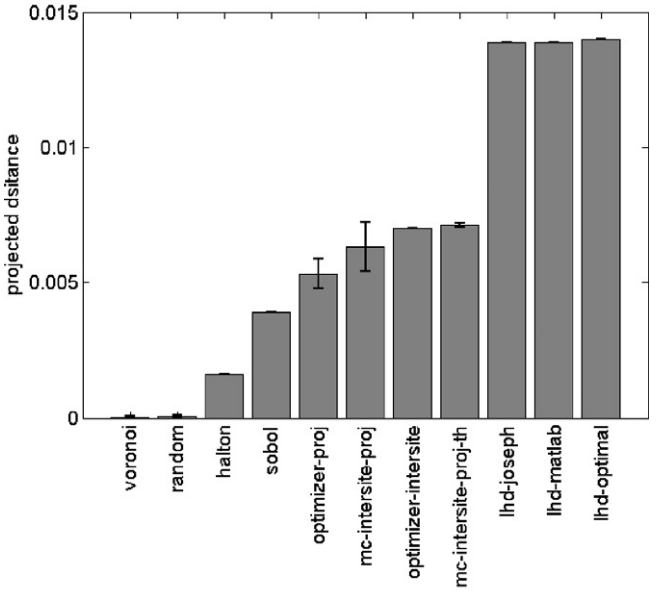
Fig. 11(b) shows the projected distance for the same designs. The `factorial` design has the worst projected distance, while the Latin hypercubes have the best score, followed by the five methods proposed in this paper, which have a projected distance about 50% worse than the Latin hypercube. This is still very good, considering that the Latin hypercube has the best possible projected distance by construction. The projected distance of many of these methods can be further improved by tweaking the algorithm parameters (such as the α threshold parameter), at the expense of intersite distance. Since the intersite distance is deemed the more important criterion of the two, more priority was given to achieving a high intersite distance in these experiments.

Fig. 12 shows the evolution over time of the intersite and projected distance for the algorithms proposed in this paper, compared to the distance scores for each `lhd-optimal` for that number of points. Note that the curve drops smoothly for all of the algorithms, except the nested Latin hypercube method. This demonstrates again the tendency of this method to get stuck in local optima, where at one point, the algorithm is forced to pick a very bad sample. The other methods suffer much less from this problem, because the points are not selected on a fixed candidate grid.

In 3D and 4D, some of the methods that were available in 2D will not work anymore. More particularly, there is no 144-point factorial design available in 3D and 4D. Also, the grid in `lhd-nested` becomes too large to evaluate completely within 15 min, so this method was also left out. Finally, computing a Delaunay triangulation becomes considerably more expensive in higher dimensions (see Crombecq et al. (2009a) for an analysis), so due to the strict time limitation, this method was left out as well.



(a) Intersite distance



(b) Projected distance

Fig. 13. The average intersite and projected distance score for each design method discussed in this paper, after generating a 144-point design in 3D.

[1] No parallelization was explicitly programmed into the algorithms, but Matlab may use different cores to execute built-on commands faster.

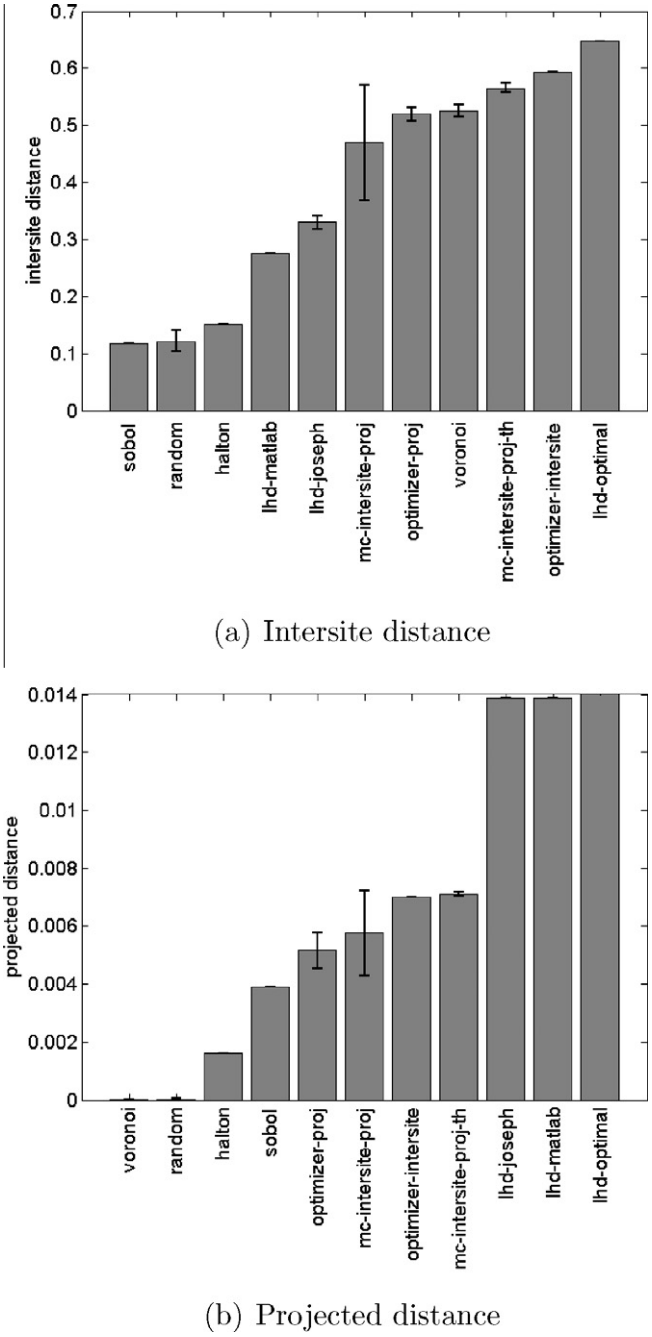


Fig. 14. The average intersite and projected distance score for each design method discussed in this paper, after generating a 144-point design in 4D.

Fig. 13 shows the intersite and projected distance scores for 3D, while Fig. 14 shows the intersite and projected distance for 4D. Note that the `optimizer-intersite` method performs 21% worse than `lhd-optimal` in 2D, but only 16% worse in 3D and 8% worse in 4D. This is an extremely good result, considering that this method only ran for 15 min, while the 4D 144-point Latin hypercube was optimized for 6 h. The projected distance is in all dimensions about 50% worse than `lhd-optimal`.

Even though a limit of 15 min was imposed on all the methods, not all methods are equally demanding in terms of computing power. Especially the Monte Carlo methods are extremely fast: the Monte Carlo designs used in this study were generated in under 5 min, as increasing the number of random points did not improve the quality of the design much. These methods also don't increase in terms of computing time when the dimension is increased. This is opposed to the optimization-based methods, which require considerably more time in higher dimensions. This may cause the `optimizer-intersite` method to become impractical in higher dimensions. The Monte Carlo method, on the other hand, should remain fast and viable in dimensions higher than 4.

7. Conclusions and future work

In this paper, several new methods for sequentially generating space-filling designs of simulation-based experiments were proposed. These methods were thoroughly compared against proven and popular techniques (such as Latin hypercubes and low-discrepancy sequences) on two criteria: intersite (or maximin) distance and projected distance. It was demonstrated that the new methods manage to generate good designs, close to the quality of a pre-optimized Latin hypercube. They also manage to generate these designs orders of magnitude faster than it takes optimizing a Latin hypercube of the same size. It was shown that in higher dimensions, the methods come even closer to the pre-optimized Latin hypercube: at 4D, the best new method produced a space-filling design only 8% worse than the pre-optimized Latin hypercube.

Of the new methods proposed in this paper, `optimizer-intersite` and `mc-intersite-proj-th` produce the best results overall. Of these, the second method is considerably faster than the first one: where the first one requires approximately 3 min to generate a design, the local optimizer utilizes the full 15 min.

As a rule of thumb, the authors suggest to use a pre-optimized Latin hypercube only if the total number of samples is known in advance. It is strongly discouraged to use the built-in Latin hypercube method from Matlab, as well as optimizing a Latin hypercube on the fly, as it may take many hours to generate a design that is as good or better than the algorithms proposed in this paper. If the total number of samples is not known in advance, or no pre-optimized Latin hypercube is available for a particular number of samples with the right number of dimensions, the first choice should be the threshold Monte Carlo method, which is easy to implement, extremely fast and performs very well in all dimensions. If a little more time can be spent on generating the design, the `optimizer-intersite` is a very good choice as well. In higher dimensions, for which optimizing a Latin hypercube can be unviable, these methods may be the only choice for producing a good space-filling design with good projective properties.

References

Audze, P., Eglais, V., 1977. New approach for planning out of experiments. Problems of Dynamics and Strengths 35, 104–107.
Batmaz, I., Tunali, S., 2003. Small response surface designs for metamodel estimation. European Journal of Operational Research 145, 455–470.
Box, G.E.P., Hunter, J.S., Hunter, W.G., 2005. Statistics for Experimenters: Design, Innovation, and Discovery. Wiley-Interscience.
Busby, D., Farmer, C.L., Iske, A., 2007. Hierarchical nonlinear approximation for experimental design and statistical data fitting. SIAM Journal on Scientific Computing 29, 49–69.
Crombecq, K., Couckuyt, I., Gorissen, D., Dhaene, T., 2009a. Space-filling sequential design strategies for adaptive surrogate modelling. In: The First International Conference on Soft Computing Technology in Civil, Structural and Environmental Engineering, 20 pages.
Crombecq, K., Gorissen, D., Tommasi, L.D., Dhaene, T., 2009b. A novel sequential design strategy for global surrogate modeling. In: Proceedings of the 41st Winter Simulation Conference, pp. 731–742.
van Dam, E.R., Husslage, B., den Hertog, D., Melissen, H., 2007. Maximin latin hypercube design in two dimensions. Operations Research 55, 158–169.
Fang, K.T., 1980. Experimental design by uniform distribution. Acta Mathematice Applicatae Sinica 3, 363–372.
Fang, K.T., Lin, D.K.J., 2003. Uniform experimental designs and their applications in industry. Handbook of Statistics 22, 131–170.

Fang, K.T., Ma, C.X., Winker, P., 2002. Centered l2-discrepancy of random sampling and latin hypercube design, and construction of uniform designs. Mathematics of Computation 71, 275–296.

Gorissen, D., Crombecq, K., Couckuyt, I., Dhaene, T., Demeester, P., 2010. A surrogate modeling and adaptive sampling toolbox for computer based design. Journal of Machine Learning Research 11, 2051–2055.

Gorissen, D., Crombecq, K., Hendrickx, W., Dhaene, T., 2007. Adaptive distributed metamodeling. High Performance Computing for Computational Science – VECPAR 2006 4395, 579–588.

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

Hickernell, F.J., 1998. A generalized discrepancy and quadrature error bound. Mathematics of Computation 67, 299–322.

Husslage, B., 2006. Maximin Designs for Computer Experiments. Ph.D. Thesis. Tilburg University, Center of Economic Research.

Jamshidi, A.A., Kirby, M.J., 2007. Towards a black box algorithm for nonlinear function approximation over high-dimensional domains. SIAM Journal on Scientific Computing 29, 941–963.

Jin, R., Chen, W., Sudjianto, A., 2002. On sequential sampling for global metamodeling in engineering design. In: Proceedings of DETC02 ASME 2002 Design Engineering Technical Conferences And Computers and Information in Engineering Conference, 10 pages.

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

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

Joseph, V.R., Hung, Y., 2008. Orthogonal-maximin latin hypercube designs. Statistica Sinica 18, 171–186.

Lehmensiek, R., Meyer, P., Müller, M., 2002. Adaptive sampling applied to multivariate, multiple output rational interpolation models with application to microwave circuits. International Journal of RF and Microwave Computer-Aided Engineering 12, 332–340.

Liu, L., 2005. Could enough samples be more important than better designs for computer experiments?. In: Proceedings of the 38th annual Symposium on Simulation, pp. 107–115.

Montgomery, D.C., 2001. Design and Analysis of Experiments.

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

Niederreiter, H., 1992. Random Number Generation and Quasi-Monte Carlo Methods. Society for Industrial and Applied Mathematics.

Owen, A.B., 1992. Orthogonal arrays for computer experiments, integration and visualization. Statistica Sinica 2, 439–452.

Qian, P.Z.G., 2009. Nested latin hypercube designs. Biometrika 96, 957–970.

Regis, R.G., 2011. Stochastic radial basis function algorithms for large-scale optimization involving expensive black-box objective and constraint functions. Computers & Operations Research 38, 837–853.

Sacks, J., Welch, W.J., Mitchell, T.J., Wynn, H.P., 1989. Design and analysis of computer experiments. Statistical Science 4, 409–435.

Simpson, T.W., Peplinski, J., Koch, P.N., Allen, J.K., 2001. Metamodels for computer-based engineering design: Survey and recommendations. Engineering with Computers 17, 129–150.

Sugiyama, M., 2006. Active learning in approximately linear regression based on conditional expectation of generalization error. Journal of Machine Learning Research 7, 141–166.

Tang, B., 1993. Orthogonal array-based latin hypercubes. Journal of the American Statistical Association 88, 1392–1397.

Viana, F.A.C., Venter, G., Balabanov, V., 2009. An algorithm for fast optimal latin hypercube design of experiments. International Journal for Numerical Methods in Engineering 82, 135–156.

Xiong, F., Xiong, Y., Chen, W., Yang, S., 2009. Optimizing latin hypercube design for sequential sampling of computer experiments. Engineering Optimization 41, 793–810.

Ye, K.Q., Li, W., Sidjianto, A., 2000. Algorithmic construction of optimal symmetric latin hypercube designs. Journal of Statistical Planning and Inference 90, 145–159.