SIAM J. SCI. COMPUT. © 2011 Society for Industrial and Applied Mathematics
Vol. 33, No. 4, pp. 1948–1974

A NOVEL HYBRID SEQUENTIAL DESIGN STRATEGY FOR GLOBAL SURROGATE MODELING OF COMPUTER EXPERIMENTS[*]

KAREL CROMBECQ[†], DIRK GORISSEN[‡], DIRK DESCHRIJVER[‡], AND TOM DHAENE[‡]

Abstract. Many complex real-world systems can be accurately modeled by simulations. However, high-fidelity simulations may take hours or even days to compute. Because this can be impractical, a surrogate model is often used to approximate the dynamic behavior of the original simulator. This model can then be used as a cheap, drop-in replacement for the simulator. Because simulations can be very expensive, the data points, which are required to build the model, must be chosen as optimally as possible. Sequential design strategies offer a huge advantage over one-shot experimental designs because they can use information gathered from previous data points in order to determine the location of new data points. Each sequential design strategy must perform a trade-off between exploration and exploitation, where the former involves selecting data points in unexplored regions of the design space, while the latter suggests adding data points in regions which were previously identified to be interesting (for example, highly nonlinear regions). In this paper, a novel hybrid sequential design strategy is proposed which uses a Monte Carlo–based approximation of a Voronoi tessellation for exploration and local linear approximations of the simulator for exploitation. The advantage of this method over other sequential design methods is that it is independent of the model type, and can therefore be used in heterogeneous modeling environments, where multiple model types are used at the same time. The new method is demonstrated on a number of test problems, showing that it is a robust, competitive, and efficient sequential design strategy.

Key words. sequential design, active learning, local linear approximation, experimental design, nonlinear function approximation

AMS subject classifications. 62L05, 62K20, 65D05, 62P30

DOI. 10.1137/090761811

1. Introduction.

In many modern engineering problems, accurate high-fidelity simulations are 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 took 36 to 160 hours to compute [16]. Because of these long computational times, the use of simulations may still be impractical for engineers who want to explore, optimize, or gain insight into the system.

In this study, the simulator is assumed to be deterministic, meaning that the same output is produced if the simulator is run twice with the same input values. Furthermore, the system under study is a grey or black box, with little or no additional

[*]Received by the editors June 11, 2009; accepted for publication (in revised form) April 13, 2011; published electronically August 16, 2011. This research was supported by FWO (Flemish Fund for Scientific Research).
http://www.siam.org/journals/sisc/33-4/76181.html

[†]COMP Research Group, Department of Mathematics and Computer Science, University of Antwerp, Middelheimlaan 1, B-2020 Antwerp, Belgium (Karel.Crombecq@ua.ac.be).

[‡]IBCN Research Group (INTEC), Ghent University-IBBT, Gaston Crommenlaan 8, B-9050 Ghent, Belgium (Dirk.Gorissen@ugent.be, Dirk.Deschrijver@ugent.be, Tom.Dhaene@ugent.be).

information available about its inner working. This means that, without running simulations, little or nothing is known about the exact behavior of the system, and no assumptions can be made about continuity or linearity or any other mathematical properties the system might have.

The goal of *global* surrogate modeling is to find an approximation function that mimics the behavior of the original system 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 surrogate model that approximates the data and the overall system behavior quite well. This is illustrated in Figure 1.1.

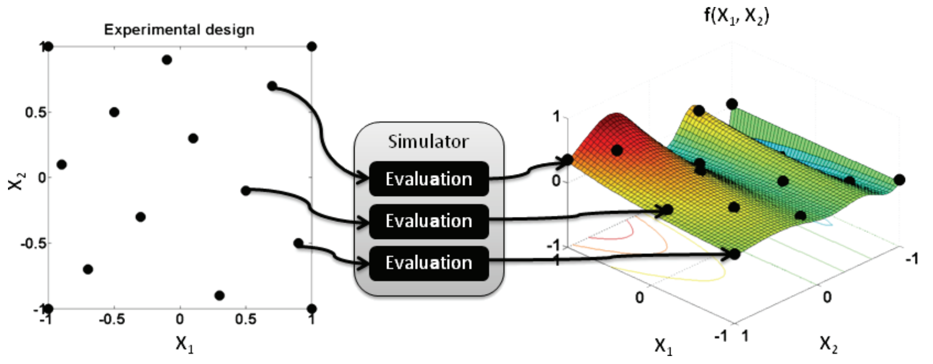


FIG. 1.1. *A set of data points is evaluated by the simulator, which outputs a response for every data point. An approximation model (global surrogate model) is fit to the data points.*

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. The local models are discarded afterwards. In *global* surrogate modeling, the goal is to create a model that approximates the behavior of the simulator on the entire domain, so that the surrogate model can then be used as a replacement for the original simulator and 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 concerned only with global surrogate modeling.

Mathematically, the simulator can be defined as an unknown function $f : \mathbb{R}^d \to \mathbb{C}$, mapping a vector of 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 $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$, 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 $\tilde{f} : \mathbb{R}^d \to \mathbb{C}$ is chosen from the (possibly) infinite 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 [2, 23].

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 nontrivial task, since little or nothing is known about this function in advance.

In this paper, we propose a new iterative scheme for selecting these data points that distributes the points in such a way that the data density is proportional to the local nonlinearity of the function, under the assumption that highly dynamic regions are more difficult to approximate than linear regions. To achieve this goal, the gradient of the function in the data points is estimated. This approach is then augmented with a space-filling technique based on Voronoi tessellations.

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 [34]. This leaves space-filling designs, which try to cover the domain as equally as possible, as the only interesting option. The advantages of classical methods are that they can be easily implemented and provide a good (and guaranteed) coverage of the domain. Examples of popular space-filling design are fractional designs [39], Latin hypercubes [38], and orthogonal arrays [8].

Sequential design (which is also known as adaptive sampling [29] or active learning [40]) further improves on this approach by transforming the one-shot algorithm into an iterative process. Sequential design methods analyze data (samples) and models 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.

2.1. Sequential design methods.

A typical sequential design method is described in Algorithm 1. First, an initial batch of data points is evaluated using a minimal experimental design. This design is usually one of the traditional designs from DOE, such as a Latin hypercube. The initial design must be large enough to guarantee a minimal coverage of the domain, but should be small enough so that there is room for improvement, allowing the sequential design strategy to do its work.

Based on the initial experimental design, a surrogate model is built and the accuracy of this model is estimated using one or more well-known error metrics. Examples of error metrics are cross-validation, an external test set, error in the data points, and so on. Based on the estimated accuracy of the model, the algorithm may (and probably will, if the initial design was small enough) decide that more samples are needed.

The locations of these additional samples are chosen by the adaptive sampling or sequential design strategy. Many different strategies are available, and the optimal strategy may depend on many factors, including the surrogate model type that is used, the number of data points, the system that is being modeled, and so on. An overview of available sequential design techniques is given later. Finally, a new surrogate model

Algorithm 1. A typical sequential design method.

```
$P \leftarrow$ initial experimental design
Calculate $f(P)$ through simulation
Train model using $P$ and $f(P)$
while accuracy not reached do
    Select new data points $P_{new}$ using sequential design strategy
    Calculate $f(P_{new})$ through simulation
    $P \leftarrow P \cup P_{new}$
    Train model using $P$ and $f(P)$
end while
```

is built using all the data gathered thus far, and the model accuracy is estimated again. If the desired accuracy is still not reached, the entire sample selection process is started all over again.

Ultimately, the goal of this scheme is to reduce the overall number of samples, since evaluating the samples (running the simulation) is the dominant cost in the entire surrogate modeling process. If data points are chosen sequentially, more information is available to base the sampling decision on compared to traditional DOE. Both the previous data points and the behavior of the intermediate surrogate model provide important information on where the next sample(s) should be located. If this additional information is used well, the total number of samples can be reduced substantially.

Note that some optimization algorithms use similar iterative schemes, but with a completely different goal. These optimization algorithms may also employ sequential design techniques to minimize the number of samples required to find the global optimum. However, they are not concerned with finding a good global approximation over the entire design space. Because of this, a lot of optimization-oriented sequential design techniques focus more on exploitation and less on exploration, and might even ignore exploration completely. In this paper, we will consider only related work on sequential design in the context of global surrogate modeling. For more information about sequential design in the context of optimization, please refer to [26, 10].

2.2. Exploration and exploitation. 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, and optima or stable regions, that have not yet 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 goal is to fill up the input domain evenly. Examples of exploration methods can be found in [33, 13].

The advantage of exploration-based sequential design methods over one-shot experimental designs is that the amount of samples evaluated depends on the feedback from previous iterations of the algorithm (when the model is accurate enough, no more samples are requested). When one large experimental design is used, too many samples may have been evaluated to achieve the desired accuracy (oversampling) or too little samples may have been evaluated (undersampling), in which case one must completely restart the experiment or resolve to sequential methods to improve the initial experimental design.

Instead of exploring the input domain, exploitation-based methods select samples 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. Examples of exploitation methods can be found in [24, 9, 41, 30, 25, 32].

In every sequential design strategy, a trade-off must be made between these two conflicting options. If a sequential design strategy focuses only on exploitation, the initial experimental design must be sufficiently large so as to capture all regions of interest right away. Otherwise, large (interesting) areas may be left unsampled, a problem which they share with optimal designs [2]. Because the simulator is often a black box, it is infeasible in practice to predict the required sample size accurately. On the other hand, if a strategy focuses only on exploration, the advantage provided by evaluating and selecting the samples sequentially is ignored, because the outputs are not used. This means that any competitive sequential design strategy must somehow accommodate both exploration and exploitation.

The necessary trade-off between exploration and exploitation can be accounted for in different ways. In sequential design methods such as those proposed in [30, 9], the trade-off is buried deep in the formulation of the algorithm. In other strategies, such as the one proposed by [41], the difference between exploration and exploitation is very clear, in that a simulated annealing approach is used to balance between the two and to switch priorities from one to the other during the modeling process.

2.3. Optimal and generic sequential design. A large class of sequential design methods assume that the model type is known in advance. This allows the algorithm to exploit the behavior of this model to guide the sampling process in the optimal direction for this specific model type. This is called optimal design. Examples of sequential design strategies using some aspects of optimal design can be found in [2, 20, 36, 28].

All of these methods incorporate to some degree information about the model in the sampling process. These sampling strategies may be highly efficient if the model for which it was developed is suitable for the problem at hand. However, this may not always be the case, as the optimal model type for a specific problem might not be known up front. This motivates the need for a generic algorithm, which makes no presumptions about the model type, the behavior of the system, or the amount of samples needed.

Generic sequential design strategies can use only the outputs from the simulator and previously built models to decide where to sample next. They cannot make any assumptions about how the model will behave or which type of model is used. In fact, completely different model types may be used at the same time in a heterogeneous modeling environment [19]. This is a major advantage over optimal sequential design strategies, especially in a black-box setting where little or nothing is known about the system in advance. In this case, choosing a model type for the problem comes down to guesswork, and if a bad choice is made, the optimal design that will be generated will not be optimal for another model type that might be tried later on. A heterogeneous modeling environment can help solve this problem by automatically looking for model types that match the problem at hand, while generating a sequential design that is not specifically tailored to one model type. Heterogeneous modeling environments, in which many completely different types of models are considered together, have a larger search space of candidate functions F, and can therefore drastically improve the accuracy of the final model.

2.4. Hybrid sequential design. We propose a novel, generic, disjunct approach, in which two different criteria are defined: one for exploration and one for exploitation. For the exploration criterion, we have chosen a Monte Carlo Voronoi approximation because of its simplicity and computational efficiency. For the exploitation criterion, we have developed an algorithm that selects additional samples near locations that deviate significantly from a local linear approximation of the system (based on the gradient of the function).

The goal of the new hybrid algorithm is to rank the neighborhood of all existing data points. This ranking is based on the two aforementioned criteria. If a neighborhood is ranked highly, it is either undersampled or very nonlinear. In either case, an additional sample will be selected in this neighborhood.

In the following sections, we will discuss the two components of the new hybrid sequential design algorithm in great detail. First, the Monte Carlo Voronoi approximation will be investigated. Next, the Local Linear Approximation (LOLA) algorithm will be discussed and analyzed. Finally, the hybrid algorithm that combines these two components will be presented and tested on a number of test cases.

3. Exploration using a Voronoi approximation. An exploration criterion must accurately and efficiently identify the region of the design space that contains the lowest density of samples. This can be done in many different ways, depending on the density measure used and on the allowed computational complexity of the algorithm.

A Voronoi tessellation (or Voronoi diagram) is an intuitive way to describe sampling density. Assume a discrete and pairwise distinct set of points $P \subset \mathbb{R}^d$ in Euclidian space, which represents the existing data points. For any point $\mathbf{p}_i \in P$, the Voronoi cell $C_i \subset \mathbb{R}^d$ contains all points in $\mathbb{R}^d$ that lie closer to $\mathbf{p}_i$ than any other point in P. The complete set of cells $\{C_1, C_2, \dots, C_n\}$ tesselate the whole space and is called the Voronoi tessellation corresponding to the set P.

To define a Voronoi tessellation more formally, we adopt the notation from [1]. For two distinct points $\mathbf{p}_i, \mathbf{p}_j \in \mathbb{R}^d$, the *dominance* of $\mathbf{p}_i$ over $\mathbf{p}_j$ is defined as the subset of the plane being at least as close to $\mathbf{p}_i$ as it is to $\mathbf{p}_j$. Formally,

$$dom(\mathbf{p}_i, \mathbf{p}_j) = \{\mathbf{p} \in \mathbb{R}^d | \, \|\mathbf{p} - \mathbf{p}_i\| \le \|\mathbf{p} - \mathbf{p}_j\|\}.$$

$dom(\mathbf{p}_i, \mathbf{p}_j)$ is a closed half plane bounded by the perpendicular bisector of $\mathbf{p}_i$ and $\mathbf{p}_j$. This bisector, which we will call the *separator* of $\mathbf{p}_i$ and $\mathbf{p}_j$, separates all points of the plane closer to $\mathbf{p}_i$ from those closer to $\mathbf{p}_j$. Finally, the Voronoi cell C_i of $\mathbf{p}_i$ is the portion of the design space that lies in all dominances of $\mathbf{p}_i$ over all of the other data points in P:

$$C_i = \bigcap_{\mathbf{p_j} \in P \setminus \mathbf{p}_i} dom(\mathbf{p}_i, \mathbf{p}_j).$$

This is illustrated in Figure 3.1. Note that in this example, the Voronoi cells are noticeably smaller near the center than near the borders; a space-filling sequential design algorithm should select new data points in the larger (darker) Voronoi cells in order to achieve a more equal distribution of data. However, the points that lie closest to the border are colored black, because their volume is infinitely high: their

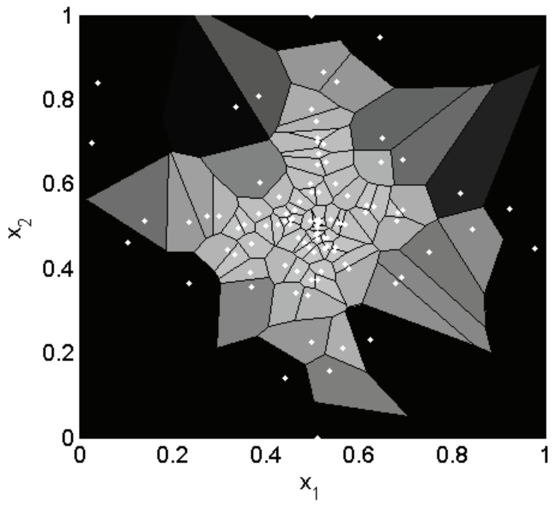


FIG. 3.1. *A set of data points and their Voronoi cells in a two-dimensional (2D) design space. Larger Voronoi cells have a darker color. The data points are drawn as white dots. Unbounded Voronoi cells are black.*

Voronoi cells reach to infinity in their respective directions. This is a basic property of Voronoi tessellations.

Computing a Voronoi tessellation is nontrivial and is usually done by calculating the dual Delaunay triangulation, from which the Voronoi tessellation can be computed in $O(n)$ time [1]. However, this still does not give us the volume of each Voronoi cell, which requires another computationally intensive step. In order to calculate the volume of each Voronoi cell, the unbounded Voronoi cells near the border of the domain must first be bounded. After this, the volume of each cell can be computed and used as a measure of density.

Fortunately, we do not need the complete computation of the Voronoi cells for our purposes: an estimation of the volume of the cells is enough. Thus, instead of solving the problem exactly by calculating the Voronoi tessellation and the volume, a Monte Carlo approach is used. In order to estimate the volume of each Voronoi cell, a large number of random, uniformly distributed test sample points are generated in the domain. For each test sample point, the minimum distance from all the data points is computed, and the test sample point is assigned to the data point which is the closest. If enough test sample points are generated like this, the algorithm produces an estimation of the (relative) size of each Voronoi cell. This is described more formally in Algorithm 2. For more information on this algorithm and its performance figures, as well as a comparison with other exploration methods, please refer to [6].

4. Exploitation using local linear approximations.

The goal of the exploitation part of a hybrid sequential design algorithm is to use the responses from previous samples to guide the sampling process to interesting regions in the design space. Which regions are deemed interesting depends mainly on the purpose of the model. In optimization, interesting regions are regions which may or do contain (local) optima. In global surrogate modeling, the goal is to find a model that accurately approximates the system over the entire domain.

However, some regions of the domain may be more difficult to approximate than

Algorithm 2. Estimating the Voronoi cell size. P is the set of data points that have to be ranked according to their respective Voronoi cell size.

```
$S \leftarrow |P| \times 100$ random points in the domain
$V \leftarrow [0, 0, \ldots, 0]$
for all $\mathbf{s} \in S$ do
  $d \leftarrow \infty$
  for all $\mathbf{p} \in P$ do
    if $\|\mathbf{p} - \mathbf{s}\| < d$ then
      $\mathbf{p}_{closest} \leftarrow \mathbf{p}$
      $d \leftarrow \|\mathbf{p} - \mathbf{s}\|$
    end if
  end for
  $V[\mathbf{p}_{closest}] \leftarrow V[\mathbf{p}_{closest}] + (1/|S|)$
end for
```

others. This may be due to discontinuities, many (local) optima close together, and so on. It is therefore intuitively a good idea to sample more densely at these "difficult" regions. More generally, samples should be distributed according to the local nonlinearity of the function. This is illustrated in Figure 4.1.

In order to be able to sample according to the local nonlinearity of the function, one needs a measure of this nonlinearity. To this end, we use the gradient of the system response. The gradient of a function $f : \mathbb{R}^d \to \mathbb{R}$ is defined as

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_d} \right). \tag{4.1}$$

The gradient of a function at a given point $\mathbf{p}_0 \in \mathbb{R}^d$ in the design space has the property that it represents the best local linear approximation for f around $\mathbf{p}_0$:

$$f(\mathbf{p}) = f(\mathbf{p}_0) + \nabla f_{\mathbf{p}_0} \cdot (\mathbf{p} - \mathbf{p}_0). \tag{4.2}$$

Therefore, the gradient can be used to estimate and quantify the nonlinearity in the region around $\mathbf{p}_0$ [21]. However, the gradient of the black-box function f is rarely

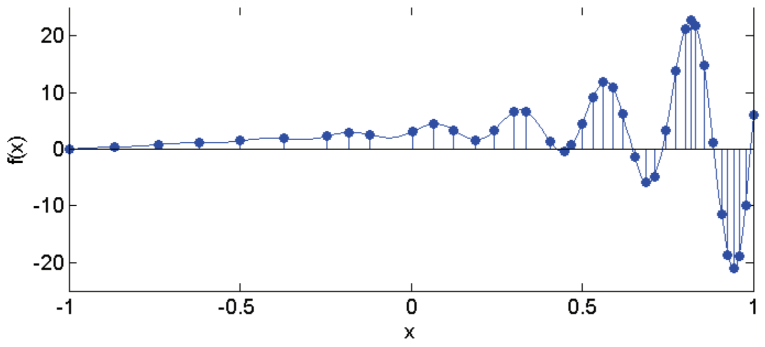


FIG. 4.1. *This plot shows a one-dimensional (1D) function and a set of samples for this function. The function is very simple and easy to approximate on the left-hand side, but very nonlinear on the right-hand side. Intuitively, more samples should be selected on the right to compensate for this nonlinearity. The samples in this plot were selected using the hybrid algorithm proposed in this paper. As expected, more samples are selected at the right-hand side to better capture the highly nonlinear behavior.*

Algorithm 3. A high level overview of the LOLA algorithm. P are all the samples that have been processed by LOLA before. P_{new} are the data points that have been selected by the sequential design algorithm in the previous iteration but have not been processed yet by the sampling algorithm. n_{new} is the number of new samples requested from the algorithm.

```
for all $\mathbf{p}_{new} \in P_{new}$ do
  for all $\mathbf{p} \in P$ do
    Try to add $\mathbf{p}_{new}$ to neighborhood $N(\mathbf{p})$ of $\mathbf{p}$
    Try to add $\mathbf{p}$ to neighborhood $N(\mathbf{p}_{new})$ of $\mathbf{p}_{new}$
    Update gradient estimations for $\mathbf{p}$ and $\mathbf{p}_{new}$
  end for
  $P \leftarrow P \cup \mathbf{p}_{new}$
end for
for all $\mathbf{p} \in P$ do
  Calculate error on gradient estimation in $N(\mathbf{p})$
end for
Pick $n_{new}$ highest ranked neighborhoods
Select new samples in these neighborhoods
```

known, so it cannot be used directly. The idea behind the Local Linear Approximation (LOLA) sampling algorithm is to estimate the gradient at the data points, in order to measure the nonlinearity around these data points. Each *region* or *neighborhood* is ranked according to its estimated nonlinearity, and new samples are selected in neighborhoods which are highly ranked. From now on, the term *neighborhood* will be used only to identify a set of samples which are chosen to represent the region around a particular data point.

A high level pseudocode overview of the algorithm can be found in Algorithm 3. When new samples have been evaluated by the simulator (for example, from a previous iteration of the sequential design method or from an initial experimental design), these samples have to be preprocessed by the LOLA algorithm. The algorithm considers a new sample $\mathbf{p}_{new}$ as a candidate neighbor sample for each previously processed sample $\mathbf{p}$. The neighborhood $N(\mathbf{p})$ of a sample will be used to estimate the gradient at $\mathbf{p}$ later on. At the same time, $\mathbf{p}$ is also considered for the neighborhood $N(\mathbf{p}_{new})$ of $\mathbf{p}_{new}$. After this initial preprocessing step, the gradient at each data point is estimated using the newly updated neighborhoods. Finally, the local nonlinearity of the neighborhoods is estimated by comparing the samples in the neighborhood to the gradient estimation. This results in a ranking of each neighborhood from linear to highly nonlinear. Finally, this ranking is used to select new samples in the highest ranking regions.

Each component of the LOLA algorithm will be discussed in greater detail in the following sections. First, the mathematical background behind the neighborhood selection algorithm will be explained. Next, it is shown how the neighborhoods can be used to estimate the gradient at the data point, and how the gradient estimation is subsequently used to estimate the local nonlinearity of the function.

4.1. Estimating the gradient. A lot of research has been done on gradient-estimating methods [11]. These methods try to estimate the gradient at one point in the design space by evaluating samples near this point. This is often done in the context of optimization, following the assumption that the gradient is a good indication of the direction of the optimum. Well-known optimization techniques such as hill climbing use this knowledge to guide the optimizer to the optimum.

These gradient estimation methods are further divided into indirect and direct estimation methods [42]. The main difference is that indirect methods assume a black-box simulator, while direct methods use internal knowledge of the simulator or its behavior. Examples of indirect gradient estimation techniques are finite differences, simultaneous perturbation, response surface methods, and frequency domain methods [12]. An overview of indirect methods can be found in [11]. Examples of direct techniques are infinitesimal perturbation analysis and likelihood ratios.

All of these methods are, however, useless for the purpose of this algorithm, as they assume that additional new samples can be evaluated in order to achieve a proper estimate for the gradient. In the context of surrogate modeling of an expensive black-box simulator, it is usually not acceptable to evaluate additional data points just to obtain the gradient. The objective is fundamentally different: in sequential design, estimating the gradient is only a small subgoal of a much larger problem, and samples are chosen so as to maximize the accuracy of the surrogate model, not the accuracy of the gradient. This renders most traditional gradient estimation methods useless for the LOLA algorithm.

The LOLA algorithm requires a method that estimates the gradient as accurately as possible using only the data that are available. No assumptions can be made about the distribution of the data over the domain, because the algorithm has no complete control over the choice of all the data points: LOLA can be used in conjunction with other (space-filling) sampling strategies, and the initial experimental design can take any form. Thus, gradient estimation methods that assume that data are available on a grid or any other pattern are unusable.

In the technique we propose, estimating the gradient in a sample location $\mathbf{p}_i$ comes down to choosing a set of neighboring samples $N(\mathbf{p}_i) = \{\mathbf{p}_{i1}, \mathbf{p}_{i2}, \dots, \mathbf{p}_{im}\}$ that lie close to the sample $\mathbf{p}_i$ and provide as much information as possible about the region around $\mathbf{p}_i$. This is illustrated in Figure 4.2 for the one-dimensional case. The sample, for which we want to find a neighborhood, is drawn as a circle. Three candidate neighbors are drawn as squares. The problem of choosing two neighbors out of these three candidates will now be considered. In Figure 4.2(a), two neighbors are chosen on opposite sides (drawn as larger squares), while in Figure 4.2(b), two neighbors are chosen on the left side.

If distance from the sample is chosen as the metric to determine the best candidates for the neighborhood, the neighborhood displayed in Figure 4.2(b) will be chosen over the one displayed in Figure 4.2(a), since both candidates on the left lie closer to the middle than the one on the right. However, it is obvious that the other neighborhood conveys much more information about the behavior of the function in this region. The information gain from adding a second neighbor close to another is much smaller than the information gain from adding one on the other side. Thus the need arises for a metric that scores neighborhoods according to how informative they are.

4.2. Constructing the neighborhoods.

When new samples are available (either from a previous iteration of the algorithm or from the initial experimental design), they have to be processed, and neighborhoods have to be assigned to each sample. This will be done in a sequential manner. Each sample will be considered as a candidate neighbor for each previously processed sample, and at the same time each previously processed sample will be considered as a candidate neighbor for the new

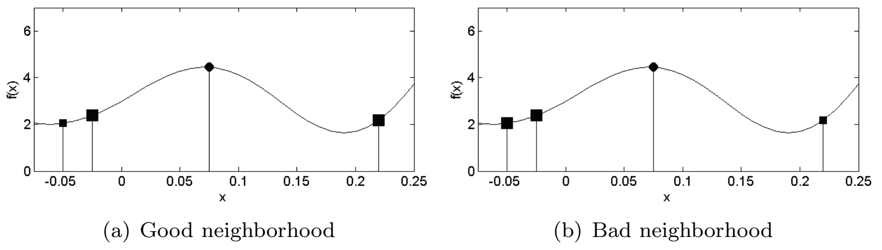


FIG. 4.2. *Two different neighborhoods of size* 2 *(*2 *samples) are shown in a* 1*D design space. The sample for which the neighborhood has been chosen is drawn as a circle in the middle. Three candidate neighbors are drawn. The two candidates which have been selected as neighbors are drawn as large squares; the third candidate, which is not in the neighborhood set, is drawn as a smaller square.*

sample. Thus, for each new sample, one needs to revisit all previous samples and their neighborhoods. We will now first consider the simplified case of finding the optimal neighborhood for one particular sample, given a set of other samples.

4.2.1. The ideal neighborhood. For clarity, we will now refer to the sample for which we want to find a neighborhood as the *reference sample* $\mathbf{p}_r \in P$ and refer to all the other available samples $P_r = P \backslash \mathbf{p}_r$ as *candidate neighbors*. Without loss of generality, we assume that $\mathbf{p}_r$ lies in the origin. This will allow us to omit the translation in the following formulas. The goal is to find a subset $N(\mathbf{p}_r) = \{\mathbf{p}_{r1}, \mathbf{p}_{r2}, \dots, \mathbf{p}_{rm}\} \subset P_r$ that best represents the region around $\mathbf{p}_r$.

The ideal neighborhood is a good representation of the region around $\mathbf{p}_r$, covering each direction equally, and thus providing the highest amount of information possible on the behavior of the function around $\mathbf{p}_r$. The samples of such a neighborhood must lie relatively close to the reference sample to be meaningful. They must also lie far away from each other in order to cover each direction as equally as possible. This results in two fundamental properties for the ideal neighborhood:

1. *Cohesion:* neighbors lie as close to the reference sample as possible.
2. *Adhesion:* neighbors lie as far away from each other as possible.

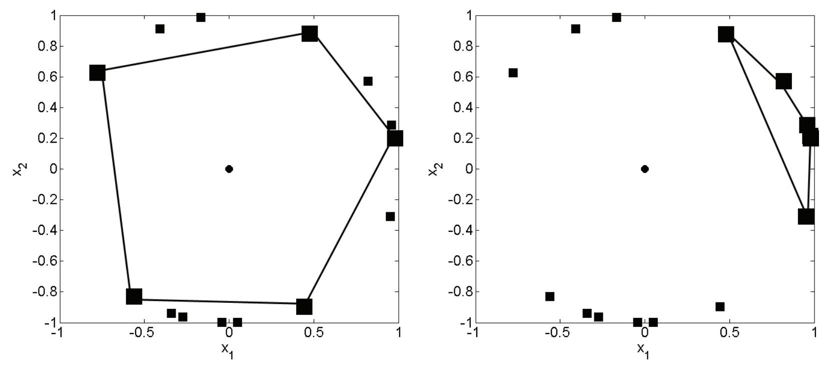
These two properties necessarily conflict with each other. The optimally cohesive neighborhood consists of all samples that lie as close to the reference sample as possible, while the optimally adhesive neighborhood consists of all samples spread out over the far reaches of the design space. Therefore, a compromise will have to be made, giving preference over adhesive neighborhoods that still lie relatively close to the reference sample.

First, the concepts of cohesion and adhesion will be defined mathematically. There are multiple sensible formulas, but the following definitions are chosen for reasons later explained. Cohesion is defined as the average distance of all neighbors from the origin (i.e., from $\mathbf{p}_r$):

$$C(N(\mathbf{p}_r)) = \frac{1}{m} \sum_{i=1}^{m} \|\mathbf{p}_{ri}\| \,. \tag{4.3}$$

Furthermore, adhesion is defined as the average minimum distance of neighbors from each other:

$$A(N(\mathbf{p}_r)) = \frac{1}{m} \sum_{i=1}^{m} \min \left\{ \|\mathbf{p}_{ri} - \mathbf{p}_{rj}\| \,|j \neq i \right\}. \tag{4.4}$$



(a) Good neighborhood (b) Bad neighborhood

Fig. 4.3. *Examples of good and bad neighborhoods chosen from the same set of candidate neighbors in a* 2*D design space. Samples that were selected for a neighborhood are drawn as large squares; rejected neighbors are drawn as small squares.*

Initially, consider the simplified case where all candidate neighbors have the same contribution to the cohesion value, which means they all lie on the same distance from the origin. This is illustrated for the two-dimensional case in Figure 4.3, where all candidate neighbors lie randomly distributed on a circle. The point in the middle is the reference sample. The goal is to find a set of m samples in the circle that maximizes the adhesion equation (4.4).

If all points on the circle are available as candidate neighbors, the best set of m neighbors are those that form an m-sided regular polygon. Of course, because of the nonuniform distribution of the samples, an ideal neighborhood can rarely be formed; however, among neighborhoods with equal cohesion values, some are clearly superior to others. There is a strict hierarchy among candidate neighborhoods, defined by their adhesion, as long as cohesion is identical for all candidate neighbors. The neighborhood with the highest adhesion value for the given candidate neighbors is illustrated in Figure 4.3(a) for $m = 5$. An example of a bad neighborhood can be found in Figure 4.3(b). This neighborhood provides no information at all on the behavior of the function on the left side of the reference sample.

In the 2D case, the ideal configuration for m neighbors when all candidates have the same cohesion contribution is, as previously mentioned, the m-sided regular polygon. In higher dimensions, this extends to the problem of placing m points in an ideal configuration on a (hyper)sphere so that the adhesion value as defined by (4.4) is maximized, which is a well-known open problem in mathematics for which their is no known general solution in all dimensions. In fact, this is considered one of the great mathematical challenges of the 21st century [5]. This is a major problem because the LOLA sampling algorithm should function independently of the dimensionality of the design space.

Because there is no all-around optimal solution for the problem of placing m points on a d-dimensional hypersphere [35], we have focused on a subproblem for which there *is* a solution known for all dimensions: the special case where $m = 2d$, or the size of the neighborhood m is twice the dimensionality d of the design space. It can be intuitively seen that the optimal configuration in one dimension has one neighbor on each side of the reference point, while the optimal configuration in two dimensions is a square. This generalizes to the d-cross-polytope [3] in the d-dimensional case.

The d-dimensional cross-polytope contains all of the points obtained by permuting the coordinates $(\pm 1, 0, 0, \ldots, 0)$. It has been proven that the cross-polytope configuration maximizes (4.4) in all dimensions [3]. This means that the ideal neighborhood resembles the cross-polytope as closely as possible. When the set of candidate neighbors consists of points that lie equally far from the origin, the best choice of neighborhood will always be a cross-polytope shape.

4.2.2. The cross-polytope ratio. In reality, the problem is more complex. Candidate points do not lie on a hypersphere; they differ in distance from the reference point. This results in a multiobjective optimization problem, in which the goal is to minimize the cohesion function defined in (4.3) while at the same time maximizing the adhesion function from (4.4). Many different methods have been proposed to solve such multiobjective optimization problems efficiently. The simplest approach is to combine the different objectives into a single aggregate objective function. This solution is acceptable only if the scale of both objectives is known, so that they can be combined into a formula that gives each objective equal weight. Fortunately, in the case of the cohesion and adhesion objectives, these weights are known.

For points lying on a sphere with a given radius, the cross-polytope is the optimal configuration, maximizing the adhesion value. This means that any given neighborhood with cohesion value $C(N(\mathbf{p}_r))$ must always have a lower adhesion value than the cross-polytope with radius $C(N(\mathbf{p}_r))$. A cross-polytope with radius $C(N(\mathbf{p}_r))$ has an adhesion value of $\sqrt{2}C(N(\mathbf{p}_r))$, because, in a cross-polytope configuration, the distance between points (the adhesion) is $\sqrt{2}$ times larger than the distance from the origin (the cohesion) for any dimension higher than 1. Hence, $\sqrt{2}C(N(\mathbf{p}_r))$ is the absolute upper bound for the adhesion value of any neighborhood with cohesion $C(N(\mathbf{p}_r))$. Based on this property, we can now describe how much a given neighborhood resembles a cross-polytope by the following measure:

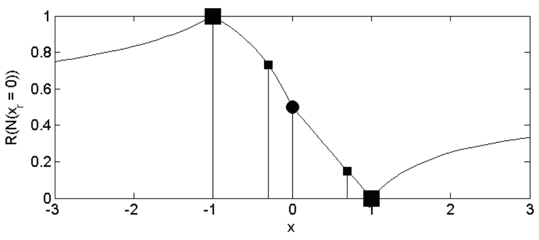
$$R(N(\mathbf{p}_r)) = \frac{A(N(\mathbf{p}_r))}{\sqrt{2}C(N(\mathbf{p}_r))}, \qquad d > 1. \tag{4.5}$$

If $R(N(\mathbf{p}_r)) = 1$ for a neighborhood, the neighborhood must form a perfect cross-polytope configuration. If the score is 0, all points of the neighborhood lie in the exact same spot, reducing the adhesion value to zero. This measure is called the *cross-polytope ratio* and indicates how much a neighborhood resembles a cross-polytope.

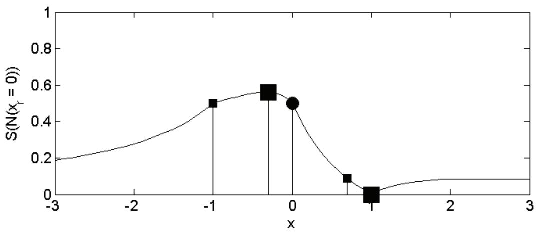
The 1D case forms a unique exception to this rule, as the distance of the two points from each other is twice the distance from the origin, instead of $\sqrt{2}$. Additionally, in the 1D case, there exists an infinite number of configurations which maximize the adhesion value for any given cohesion value c: any two points x, $-2c+x$ with $0 \leq x \leq c$ will result in a maximized adhesion value, since $|(-2c+x) - x| = 2c$. So, we propose an alternative measure, which in the 1D case exhibits behavior similar to (4.5) in the d-dimensional case ($d > 1$):

$$R(N(\mathbf{p}_r)) = 1 - \frac{|\mathbf{p}_{r1}+\mathbf{p}_{r2}|}{|\mathbf{p}_{r1}|+|\mathbf{p}_{r2}|+|\mathbf{p}_{r1}-\mathbf{p}_{r2}|}, \qquad d = 1. \tag{4.6}$$

To illustrate the behavior of the cross-polytope ratio, we now consider the case of finding the optimal neighborhood for a sample $\mathbf{p}_r = 0$ in a 1D design space. Assume that one sample $\mathbf{p}_{r1} = 1$ is already added to the neighborhood of p_r. The cross-polytope ratio is illustrated in Figure 4.4(a). This plot shows the cross-polytope ratio when the second neighbor is moved over the domain while the first neighbor is kept fixed at 1. As expected, the function is maximized at location -1, because this will



(a) Cross-polytope ratio



(b) Neighborhood score

FIG. 4.4. *Cross-polytope ratio and neighborhood score for reference sample* 0*, with one neighbor fixed at* 1 *in a* 1*D design space. Three other candidate neighbors are drawn as small squares. The candidates with, respectively, the highest cross-polytope ratio and neighborhood score are drawn as large squares.*

result in a perfect cross-polytope neighborhood. For any positive value x, the score at $-x$ is always better than the one at x, which illustrates that samples that lie on the opposite side of the fixed neighbor are preferred, because they add more information.

The cross-polytope ratio has some useful and desirable properties: sampling on the "unsampled" side is highly encouraged, and samples near a cross-polytope configuration are preferred over samples far away. However, this metric has a serious drawback, as it completely ignores distance from the reference point when scoring neighborhoods.

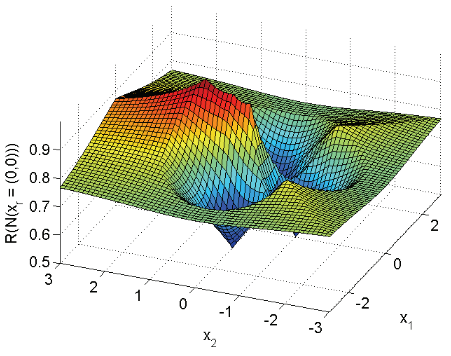
This is also illustrated in Figure 4.4(a), where four candidate neighbors are visualized as squares. The two big squares are the ones that are selected as neighbors, because they form a perfect cross-polytope. However, it is clear that the two points that lie closer to the origin form the best neighborhood and convey much more information about the environment of the reference sample $\mathbf{p}_r = 0$ than the two that were selected. Because of this, the cross-polytope ratio cannot be used directly as a measure for selecting a suitable neighborhood. In order to resolve this issue, the distance of the candidate neighbors from the origin must be taken into account.

4.2.3. The neighborhood score. We define the *neighborhood score* as follows:

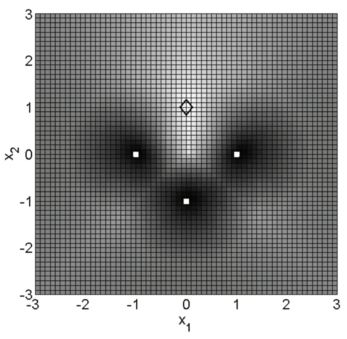
$$S(N(\mathbf{p}_r)) = \frac{R(N(\mathbf{p}_r))}{C(N(\mathbf{p}_r))}. \tag{4.7}$$

By dividing the cross-polytope ratio by the cohesion value $C(N(\mathbf{p}_r))$ of the neighborhood, a measure is acquired that prefers neighborhoods that resemble a cross-polytope as well as neighborhoods that lie closer to the reference sample $\mathbf{p}_r$.

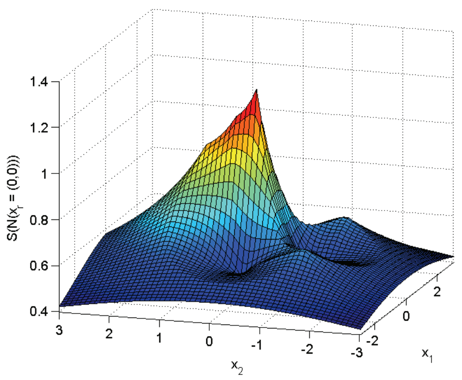
The neighborhood score is shown in Figure 4.4(b) for the 1D case. By using the neighborhood score (instead of the cross-polytope ratio), samples lying closer to



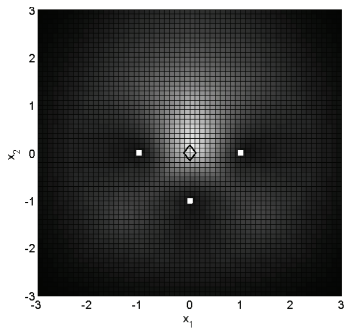
(a) Cross-polytope ratio.



(b) Cross-polytope ratio surface plot.



(c) Neighborhood score.



(d) Neighborhood score surface plot.

FIG. 4.5. *The cross-polytope ratio and the neighborhood score in a* 2*D design space with three neighbors fixed at* $(-1,0),(1,0),(0,-1)$*. The fourth neighbor is moved over the domain, and the resulting cross-polytope ratio and neighborhood score are displayed in* (a) *and* (c)*, respectively. The global maximum is indicated by a diamond on the surface plots, while the fixed neighbors are drawn as squares.*

the origin are given preference over samples that lie farther away. However, the key properties of the cross-polytope ratio are maintained: samples on the unsampled side are still preferred over samples near other neighbors. In Figure 4.4(b), the sample closer to 0 is now chosen as the second neighbor instead of the sample at -1 as in Figure 4.4(a).

In Figure 4.5, the cross-polytope ratio and the neighborhood score are shown for the 2D case. In the case of the cross-polytope ratio (Figure 4.5(a)), the surface is maximized at $(0,1)$. In the case of the neighborhood score function (Figure 4.5(c)), the surface is maximized at $(0,0)$. In both cases, the surface reaches a local minimum at the three fixed neighbors, which makes sense, since adding another point at the same location will not provide any additional information. Furthermore, both functions strongly prefer new samples around $(0,1)$. It is clear that in the 2D case the neighborhood score shows behavior similar to the 1D case shown in Figure 4.4. Because the fundamental properties of the cross-polytope do not depend on its dimensionality, the neighborhood score behaves the same in higher dimensions.

4.3. Gradient estimation. Based on the neighborhood score (as defined in section 4.2.3), we can select a good set of neighbors for each reference sample $\mathbf{p}_r$ so that the neighborhood provides a proper coverage of the design space in each direction. These neighbors may not be the samples closest to $\mathbf{p}_r$, but they provide more information about the behavior of the system around the reference sample than other neighborhoods with potentially lower cohesion values.

Once a set of suitable candidate neighbors has been chosen, estimating the gradient becomes straightforward. We define the neighborhood for reference sample $\mathbf{p}_r$ as $N(\mathbf{p}_r) = \{\mathbf{p}_{r1}, \mathbf{p}_{r2}, \ldots, \mathbf{p}_{rm}\}$, with $m = 2d$, as explained earlier. The gradient at $\mathbf{p}_r$ is estimated by fitting a hyperplane through $\mathbf{p}_r$ and its neighbors. To ensure that the hyperplane goes exactly through $\mathbf{p}_r$, the following system is solved using least squares:

$$\begin{pmatrix} p_{r1}^{(1)} - p_r^{(1)} & p_{r1}^{(2)} - p_r^{(2)} & \ldots & p_{r1}^{(d)} - p_r^{(d)} \\ p_{r2}^{(1)} - p_r^{(1)} & p_{r2}^{(2)} - p_r^{(2)} & \ldots & p_{r2}^{(d)} - p_r^{(d)} \\ \vdots & \vdots & & \vdots \\ p_{rm}^{(1)} - p_r^{(1)} & p_{rm}^{(2)} - p_r^{(2)} & \ldots & p_{rm}^{(d)} - p_r^{(d)} \end{pmatrix} \begin{pmatrix} g_r^{(1)} \\ g_r^{(2)} \\ \vdots \\ g_r^{(d)} \end{pmatrix} = \begin{pmatrix} f(p_{r1}) \\ f(p_{r2}) \\ \vdots \\ f(p_{rm}) \end{pmatrix}, \tag{4.8}$$

where $\mathbf{p}_{ri} = (p_{ri}^{(1)}, p_{ri}^{(2)}, \ldots, p_{ri}^{(d)})$ is the ith neighbor for $\mathbf{p}_r$ with evaluated value $f(\mathbf{p}_{ri})$, and $\mathbf{g} = (g_r^{(1)}, g_r^{(2)}, \ldots, g_r^{(d)})$ is the gradient that is being calculated. This system can be inverted and will result in the hyperplane which minimizes the distance from the neighbors in a least squares sense. This results in the best linear approximation for the neighbors, which will converge to the best local linear approximation at the reference sample as the neighbors lie closer to the reference sample.

Because of the way the neighbors p_{ri} are chosen, the matrix from (4.8) is always well-conditioned. This can be seen from the fact that, in a perfect cross-polytope configuration, all the vectors $\mathbf{p}_{ri} - \mathbf{p}_r$ are orthogonal with respect to each other, and therefore the matrix composed of these vectors is well-conditioned. Because the neighborhood selection algorithm produces a neighborhood that resembles a cross-polytope as closely as possible, the resulting matrix is also well-conditioned.

It is very important to have a good neighborhood in order to get a good estimation of the gradient. If all neighbors lie in the same direction, it becomes impossible to make a good estimation of the gradient, since the hyperplane will be completely biased towards the behavior of the system near the neighbors. This is why a lot of attention is paid to proper neighborhood selection in the LOLA algorithm.

4.4. Nonlinearity measure. Once the gradient estimation is available, it is possible to estimate the (non)linearity of the system behavior around the reference sample. The local nonlinearity of the system can be estimated from the normal of the hyperplane using the following formula:

$$E(\mathbf{p}_r) = \sum_{i=1}^{m} \left| f(\mathbf{p}_{ri}) - (f(\mathbf{p}_r) + \mathbf{g} \cdot (\mathbf{p}_{ri} - \mathbf{p}_r)) \right| . \tag{4.9}$$

This formula computes how much the response at the neighbors differs from the local linear approximation that was computed earlier. The nonlinearity measure $E(\mathbf{p}_r)$ can now be used to get an idea of how nonlinearly the function behaves in the area around the reference sample $\mathbf{p}_r$, using solely previously evaluated samples to compute this estimation. Furthermore, this approach works both for real outputs and for complex outputs, making the LOLA algorithm suitable in both cases without requiring a change in the algorithm. Also note that this is the only place in the algorithm where the actual simulator output is used; the rest of the algorithm works on input values only and, if desired, some ratios and scores can be precalculated, independent of the actual system behavior.

5. Hybrid sequential design using Voronoi approximation and LOLA. In the previous two sections, we have basically developed two different methods for ranking previously evaluated samples. The Monte Carlo Voronoi approximation method explained in section 3 ranks samples according to the size of their Voronoi cells, while the LOLA algorithm discussed in section 4 ranks samples according to local nonlinearity. The first method is an exploration strategy, while the second is an exploitation strategy. By combining these two metrics, we can counteract the disadvantages of both approaches and deliver a solid, robust, and flexible sampling strategy. From now on, this hybrid sampling strategy will be referred to as *LOLA–Voronoi*.

In order to properly combine the two measures, they first have to be normalized. The Voronoi cell size is already in the range $[0, 1]$, because it represents which portion of the design space is contained within each Voronoi cell. The nonlinearity measure, however, is initially not scaled to $[0, 1]$. Therefore, the hybrid score for a sample $\mathbf{p}_i \in P$ is computed using the following formula:

$$H(\mathbf{p}_i) = V(\mathbf{p}_i) + \frac{E(\mathbf{p}_i)}{\sum_{j=1}^{n} E(\mathbf{p}_j)}. \tag{5.1}$$

The LOLA–Voronoi sequential design strategy is described in pseudocode in Algorithm 4. First, the nonlinearity measure $E(\mathbf{p})$ is calculated using the LOLA algorithm. Then, the Voronoi cell size $V(\mathbf{p})$ is computed using the Voronoi approximation, as defined in Algorithm 2. These two measures are then combined into a single value, and this value is used to rank all the samples according to how undersampled their environment is. Finally, n_{new} new samples are selected around the samples which are ranked the highest. This is done by generating a number of random points in the Voronoi cell of $\mathbf{p}_i$ and picking the one farthest away from $\mathbf{p}_i$ and its neighbors. This process can be sped up by reusing the points that were generated for the Voronoi approximation. Combining an exploration strategy with an exploitation strategy guarantees that the design space is filled up everywhere and that no large areas are left unsampled. However, nonlinear regions will be sampled much more densely, which will in turn allow the surrogate model to capture complex behavior more easily.

Algorithm 4. Hybrid sequential design using Voronoi approximations and LOLA. n_{new} is the number of samples requested by the user of the algorithm.

```
for all $\mathbf{p} \in P$ do
  Calculate $E(\mathbf{p})$
  Calculate $V(\mathbf{p})$
  Compute final ranking $H(\mathbf{p})$ using $E(\mathbf{p})$ and $V(\mathbf{p})$
end for
Sort $P$ by $H$
for $i = 1$ to $n_{new}$ do
  $\mathbf{p}_{new} \leftarrow$ location near $\mathbf{p}_i$ farthest from other samples
  $P_{new} \leftarrow P_{new} \cup \mathbf{p}_{new}$
end for
```

Note that the choice of n_{new} affects the quality of the design that is being generated: if n_{new} is smaller, more information is available to determine the location of the next sample as optimally as possible. Preferably, n_{new} should be set to 1, but higher values will also produce good designs, because at most one sample is chosen in each Voronoi cell during each iteration of the sequential algorithm, thereby ensuring a proper coverage of the design space, even for $n_{new} > 1$.

The efficiency of the LOLA–Voronoi sequential design strategy comes at the cost of additional computing time for sample selection, which is mainly caused by the neighborhood selection algorithm, which considers each sample as a candidate neighbor for every other sample. Every time a new sample is evaluated, it has to be considered as a potential candidate for every other sample. This is done by replacing each current neighbor by the new candidate and calculating the neighborhood score. Finally, the neighborhood is picked that has the best score. The standard algorithm runs in $O(n^2)$ time, where n is the number of samples evaluated so far. Whether this is an issue or not depends largely on the problem at hand. If evaluating samples is very expensive (costing minutes, hours, or days), the additional computing time for the sequential sampling process may be negligible compared to the sample evaluation time. However, if acquiring new data is relatively cheap and the number of data points is large, the neighborhood selection algorithm can severely slow down the overall modeling process.

Fortunately, many heuristics can be applied to the neighborhood selection algorithm to significantly speed up the entire process. For example, samples that lie much farther away from the reference sample than the currently selected neighbors can be rejected right away, without considering them as candidate neighbors, and thus saving the expensive neighborhood score calculation step. This simple modification can reduce the number of neighborhood score calculations by 90%, without hurting the quality of the design that is generated by the algorithm. Speeding up LOLA–Voronoi by means of heuristics is a subject of ongoing research and will be explored and discussed in future work. The current implementation of LOLA–Voronoi, which was used in the experiments for this paper, already works extremely quickly up to several thousands of samples, which should be generally enough for typical problems.

6. Experimental setup.

In previous studies, LOLA–Voronoi has repeatedly proven its excellent performance on problems from different research domains, such as electrical engineering [19, 17, 14, 7], aerospace engineering [14], and hydrology [4]. In this section, three examples are discussed, each demonstrating the flexibility and robustness of LOLA–Voronoi under different conditions. LOLA–Voronoi will be compared against a number of methods that have proven their merit in other studies. First, a space-filling sequential design method will be included, which is in fact the Voronoi component of LOLA–Voronoi as described in section 3, and which has demonstrated its efficiency compared to other space-filling design strategies in [6].

Additionally, an exploitation-based method will be included that calculates the difference between the models constructed in previous iterations and selects samples where the models disagree the most. This is done by constructing a very dense, randomly perturbated grid over the entire design space (typically 2500 points, even though the number may be higher for high-dimensional problems). The method then evaluates and compares the best models from the previous iterations on this grid. This is done by subtracting the outputs from these models pairwise from each other and finding the locations on the grid where the difference is greatest, as described by [22]. Locations where the models disagree indicate locations of high uncertainty

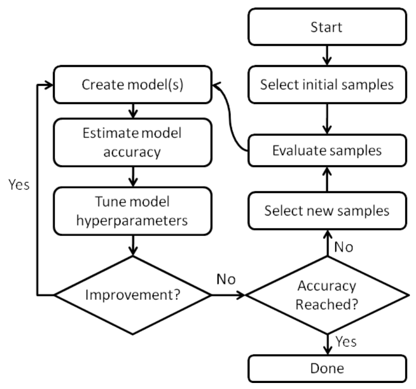


FIG. 6.1. *Flow-chart of the SUMO Toolbox.*

and will be sampled next. Because surrogate models can be evaluated fairly quickly, evaluating them over such a dense grid is usually not a problem. From now on we will refer to this strategy as model error sampling, because it estimates the approximation error by subtracting the model outputs from each other. Finally, random sampling was also included as a base case.

6.1. SUMO research platform. In order to compare the different sampling strategies, each method was implemented in the SUrrogate MOdeling (SUMO) research platform [15, 17]. This MATLAB toolbox, designed for adaptive surrogate modeling and sampling, has excellent extensibility, making it possible for the user to add, customize, and replace any component of the sampling and modeling process. It also has a wide variety of built-in test functions and test cases, as well as support for many different model types. Because of this, SUMO was a good choice for conducting this experiment.[1]

The work-flow of SUMO is illustrated in Figure 6.1. First, an initial design (typically a sparse Latin hypercube or a fractional design) is generated and evaluated. Then, a set of surrogate models is built, and the accuracy of these models is estimated using a set of measures (for example, cross-validation or an external validation test set). Each model type has several hyperparameters which can be modified, such as the order of numerator and denominator for rational models, number and size of hidden layers in neural networks, smoothness parameters for radial basis function (RBF) models, and so on. These parameters are adjusted using a hyperparameter optimization technique, and more models are built until no further improvement can be made by changing the hyperparameters. If the overall desired accuracy has not yet been reached, a call is made to the sequential design routine, which selects a new sample to be evaluated, and the algorithm starts all over again. For more information on the different components of the SUMO Toolbox, please refer to [17, 14].

We used a very sparse Latin hypercube (10 samples) augmented with a 2-level fractional design as the initial design for the SUMO Toolbox. Because we want to measure the efficiency of a sequential design strategy, the initial design is kept very small, so that the majority of the samples are chosen adaptively.

The quality of the model is measured by comparing the model against a very

[1]The SUMO Toolbox v7.0 (including all the algorithms described in this paper) can be downloaded from http://www.sumo.intec.ugent.be, allowing for a full reproduction of the experiments.

dense, pre-evaluated test set. Thus, the error is a very accurate estimate of the true prediction error of the model. Each run will be terminated when the root mean square error (RMSE) of the model is below 0.05. The RMSE is defined as

$$\text{RMSE}(f,\tilde{f}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left|f(\mathbf{q}_i) - \tilde{f}(\mathbf{q}_i)\right|^2}, \tag{6.1}$$

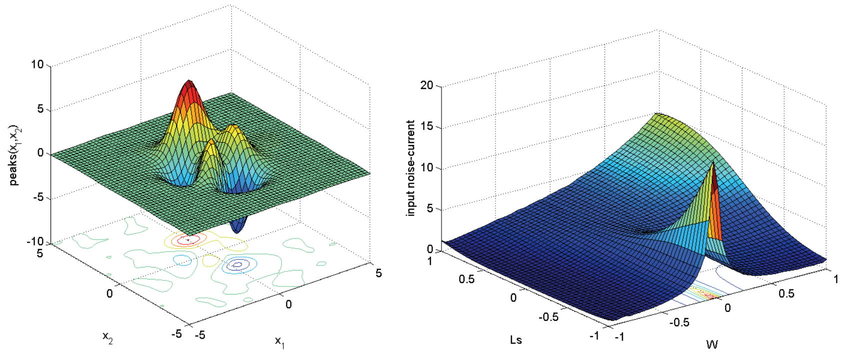
where f is the target function, $\tilde{f}$ is the surrogate model, and $\mathbf{q}_i$ are the samples in the dense pre-evaluated test set. At the end of each run, the number of samples required to reach this accuracy will be recorded. To take into account noise caused by random factors in the SUMO Toolbox (such as randomization in the model parameter optimization algorithm), the configuration for each sampling strategy will be run 10 times, and the average will be used for the results.

6.2. Test cases. Three test cases will be examined in this study, each demonstrating the quality and robustness of LOLA–Voronoi in a different context.

6.2.1. Case 1: Peaks function. The first test problem is a 2D function called `Peaks`, which is available in MATLAB as a built-in command. The `Peaks` function is obtained by translating and scaling Gaussian distributions. It is interesting to note that the function is almost zero on the entire domain except for the region close to the origin, where it has several local optima in close proximity. In order to demonstrate the ability of LOLA–Voronoi to zoom in on nonlinear regions, the problem will be modeled on three different domains $[-3,3]^2$, $[-5,5]^2$, and $[-8,8]^2$. We expect that the advantage of LOLA–Voronoi over the other methods will substantially increase as the domain grows, because the larger domains will contain proportionally more flat space. The `Peaks` function for $[-5,5]^2$ is illustrated in Figure 6.2(a).

Because the `Peaks` function is a combination of Gaussian distributions, Kriging is a natural choice for modeling this function. Kriging originates from geostatistics and was introduced as part of a modeling technique called Design and Analysis of Computer Experiments (DACE) [34]. In Kriging, the surrogate model is of the form

$$\tilde{f}(x) = \sum_{i=1}^{n} \beta_i h_i(\mathbf{x}) + Z(\mathbf{x}). \tag{6.2}$$



(a) `Peaks` (b) LNA

FIG. 6.2. *The first two test problems.*

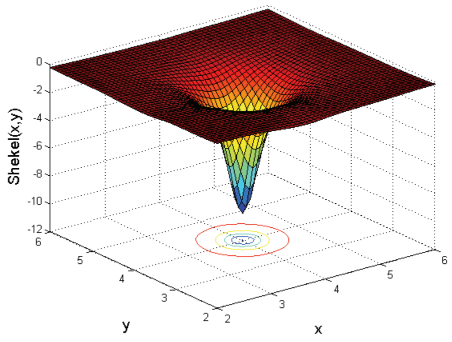
The first part of the equation is a linear regression model, while the second part $Z(\mathbf{x})$ is a Gaussian random process with zero mean and nonzero covariance. Because the correlation function for the random process $Z(\mathbf{x})$ is taken to be Gaussian, the `Peaks` function can be modeled accurately. However, the convergence rate of the modeling process greatly depends on the sampling strategy. It is expected that a sampling strategy which focuses extensively on the highly dynamic area near the origin will converge faster than a strategy which samples the design space uniformly, since many samples will be wasted on the flat regions near the edges of the design space. After each sequential step, the hyperparameters of the Kriging model are optimized dynamically.

6.2.2. Case 2: Low-noise amplifier. The second test problem is a test case from electronics: a narrowband Low Noise Amplifier (LNA), which is a simple RF (radio frequency) circuit [27]. An LNA is the typical first stage of a receiver, having the main function of providing the gain needed to suppress the noise of subsequent stages, such as a mixer. In addition it has to give negligible distortion to the signal while adding as little noise as possible itself. The performance figures of an LNA (gain, input impedance, noise figure, and power consumption) can be determined by means of rigorous computer simulations where the underlying physical behavior is accurately taken into account. From these performance figures, the input noise-current $\sqrt{\overline{i_{in}^2}}$, which is shown in Figure 6.2(b), was chosen because it is the most challenging performance figure of the LNA [18]. The input parameters are the inductance L_s and the MOSFET width W. The input space is scaled to $[-1, 1]$ for practical reasons. Note that the response is very linear in most of the design space, except for one very tall ridge where $W = 0$, which makes it an excellent test case for LOLA–Voronoi. For more information on this problem, please refer to [18].

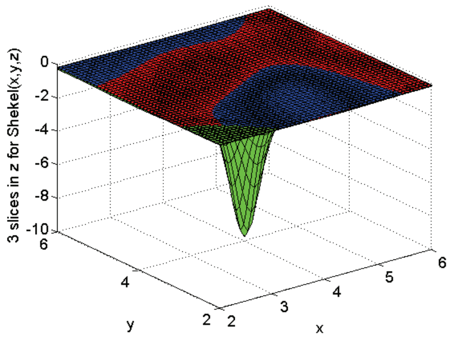
To demonstrate the efficiency of LOLA–Voronoi in suboptimal conditions, the LNA problem will be modeled with three different model types. The first model type is artificial neural networks (ANN), because they have proven to be the best choice for this test case in related studies [17]. The ANN models are based on the MATLAB Neural Network Toolbox and are trained with Levenberg Marquard backpropagation with Bayesian regularization [31] (300 epochs). The topology and initial weights are optimized by a genetic algorithm. Furthermore, the LNA problem will also be modeled with RBF models and rational models. In preliminary experiments, both RBF models and rational models had more difficulty modeling the LNA problem than ANN. The goal is to demonstrate that, even with a suboptimal pairing of model and problem, LOLA–Voronoi should produce better results than uniform sampling.

6.2.3. Case 3: Shekel function. The third and final test case is the `Shekel` function, which is a well-known multidimensional test function from optimization [37]. We use the four-dimensional (4D) version on a $[2, 6]^4$ domain, with a global optimum at $(4, 4, 4, 4)$. In order to demonstrate the scalability of LOLA–Voronoi to higher dimensions, this problem was modeled in two, three and four dimensions. For the 2D case, the last two inputs were fixed at 4, while in three dimensions only the last input is fixed at 4. The 2D and 3D versions of the function are shown in Figure 6.3. Note that only one nonlinear region exists around $(4, 4, 4, 4)$. In the 3D case, for other values of z besides 4, the function remains mostly zero.

This function was modeled in all dimensions with artificial neural networks, because of their good overall performance and robustness. Because of the simple nature of the surface, it is expected that, even for higher dimensions, it should be relatively easy to model this problem using ANN.



(a) `Shekel` 2D



(b) `Shekel` 3D

Fig. 6.3. *The 2D and 3D versions of the `Shekel` function modeled in this experiment.*

6.3. Results. The results of the experiments are depicted in Table 6.1.

For the `Peaks` function, LOLA–Voronoi produced the best results in every test case, performing 15% better than Voronoi-based sampling, 29% better than model error sampling, and 36% better than random sampling on the $[-3, 3]$ domain. Voronoi-based uniform sampling performs much better than random sampling because the sample size is relatively small, resulting in large unsampled regions for random sampling, while Voronoi-based sampling properly covers up the design space quite uniformly. The model error method, even though it is an exploitation-based method, performs worse than space-filling sampling using Voronoi, but is still better than random sampling. On the $[-5, 5]$ and $[-8, 8]$ domains, LOLA–Voronoi requires only a small number of additional samples to fill up the flat regions, while Voronoi-based, model error and random sampling both waste large amounts of samples on these regions, resulting in a huge difference in the total number of samples compared to LOLA–Voronoi. On the $[-8, 8]$ domain, Voronoi and model error sampling require,

Table 6.1

Summary of the test results of modeling `Peaks`, LNA, and `Shekel` with different sampling strategies. The average number of samples required to reach an RMSE error of 0.05, *and the standard deviation (over* 10 *runs) are shown for each sampling strategy.*

Sampling strategy	Peaks $[-3, 3]$	Peaks $[-5, 5]$	Peaks $[-8, 8]$
LOLA–Voronoi	**90 ± 0**	**114 ± 4**	**135 ± 16**
Voronoi	106 ± 6	247 ± 7	516 ± 26
Model error	126 ± 9	275 ± 31	648 ± 33
Random	141 ± 14	355 ± 92	720 ± 190
	LNA ANN	LNA RBF	LNA rational
LOLA–Voronoi	**95 ± 2**	**183 ± 19**	**131 ± 26**
Voronoi	173 ± 8	> 1500 *	1112 ± 714
Model error	435 ± 74	> 1500 *	1048 ± 465
Random	263 ± 140	> 1500 *	1567 ± 756
	Shekel-2D	Shekel-3D	Shekel-4D
LOLA–Voronoi	81 ± 12	**195 ± 74**	**204 ± 69**
Voronoi	122 ± 16	448 ± 107	543 ± 103
Model error	**72 ± 6**	199 ± 51	212 ± 63
Random	134 ± 26	511 ± 158	557 ± 274

* The RBF implementation in the SUMO Toolbox can only generate models up to 1500 data points, due to memory limitations in MATLAB. Since the accuracy was not reached at this point, the run was halted.

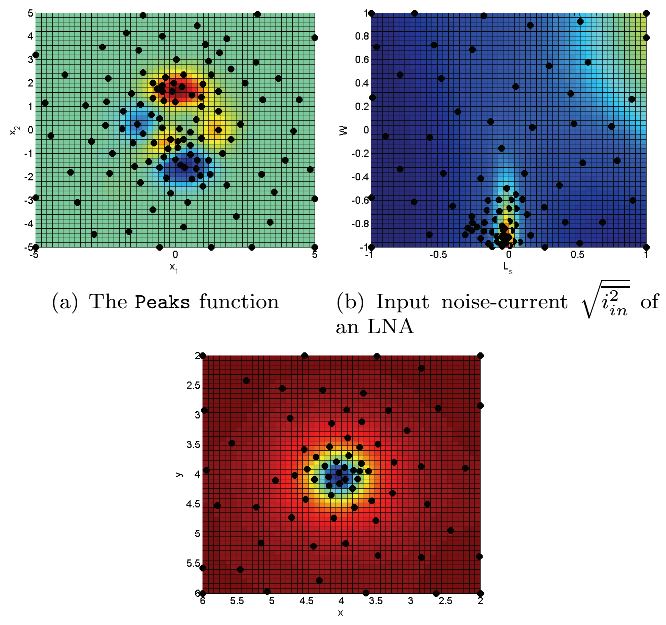
respectively, about 3 and 5 times the number of samples that LOLA–Voronoi needs to achieve the same accuracy.

For the LNA test case, a considerable improvement can be noted as well. Because there is only one small nonlinear region, focusing heavily on this region greatly improves the accuracy. LOLA–Voronoi quickly identifies this nonlinear region and selects additional samples nearby. This allows LOLA–Voronoi to reach the same accuracy as Voronoi with only half the number of samples in the ANN case. In this test case, model error sampling needs considerably more samples than random sampling. This highlights an important issue with the model error method: its inherent instability. Model error sampling tends to cluster points in locations that are difficult to approximate by the model type used, leaving large portions of the design space undersampled and unexplored. If the initial experimental design does not have any samples located on the ridge, the model error method will focus on improving (relatively) uninteresting regions, instead of exploring the design space to locate this ridge. This can severely reduce the performance of the method.

For the other model types, the difference is even more dramatic: for RBF models, LOLA–Voronoi needs only 183 samples, while the other methods fail to achieve an accuracy of 0.05 at all before reaching 1500 samples, at which point the SUMO Toolbox was aborted due to memory limitations. It is expected that, eventually, an accuracy of 0.05 can be reached, but this might take thousands of samples. For rational models, LOLA–Voronoi needs only 131 samples, while the other methods require an order of magnitude more. This example highlights the importance of focusing on difficult regions of the design space. Please note again that these results were obtained by running the toolbox 10 times for each configuration, thereby ruling out potential lucky runs.

Finally, the `Shekel` function demonstrates that LOLA–Voronoi works just as well in higher dimensions. For this test case, model error sampling shows that under the right circumstances, it can produce very good results, obtaining an average number of samples marginally lower than LOLA–Voronoi for the 2D case, and marginally higher for the 3D and 4D cases. These two methods perform considerably better in two, three, and four dimensions than Voronoi-based and random sampling. In the 4D case, LOLA–Voronoi needs less than half of the samples that Voronoi needs, and little more than 1/3 of the samples random sampling needs. The `Shekel` function has only one nonlinear area near the middle of the design space. In higher dimensions, the part of the design space that is completely linear is much larger than in lower dimensions. Therefore, the gap between exploitation-based and exploration-based methods is much larger in higher dimensions. Even though model error sampling performs very well for this last test case, the previous two cases show that it is a very unstable method, which can do very well and very poorly, and its performance is largely dependent on the problem at hand and the model type used. LOLA–Voronoi, on the other hand, works very well for all the test cases, for different model types and for multiple dimensions, due to its robust implementation of exploration and exploitation.

To illustrate how LOLA–Voronoi identifies nonlinear regions while still maintaining proper domain coverage, one run with the LOLA–Voronoi algorithm for each test problem is shown in Figure 6.4. Figure 6.4(a) contains all the points that were selected by LOLA–Voronoi during one of the runs for the `Peaks` problem on the $[-5, 5]$ domain, while Figure 6.4(b) contains the samples selected during one run of the LNA problem. Finally, Figure 6.4(c) shows one run for the 2D version of the `Shekel` prob-



(a) The `Peaks` function (b) Input noise-current $\sqrt{i_{in}^2}$ of an LNA

(c) `Shekel` 2D

FIG. 6.4. *The sample distribution of one run of the first two test cases using the LOLA–Voronoi hybrid sampling strategy.*

lem. It is clear that the hybrid strategy efficiently located the nonlinear regions and focused sampling heavily on those regions, without completely neglecting the other parts of the design space. In Figure 6.4(a), the flat region near the edges is sampled sparsely, but the samples are distributed quite evenly over the entire flat region. The steep slopes in the middle are sampled much more densely than gentler slopes, which are still sampled more densely than the flat regions. In Figure 6.4(b), the tall ridge is sampled much more densely than the rest of the design space. Due to this intelligent sampling approach, the average number of samples required is reduced drastically, potentially saving lots of resources and time.

7. Conclusion.

In this paper, we proposed a novel hybrid sequential design technique that combines an exploration metric based on Voronoi tessellations with an exploitation metric using local linear approximations. We showed that LOLA–Voronoi performs better than the model error–based, Voronoi-based, and random sampling methods in a number of different test cases, thus demonstrating the usefulness of hybrid sequential design methods. It was shown that LOLA–Voronoi outperforms the other methods for different model types and problems and in multiple dimensions. LOLA–Voronoi was also successfully applied to multiple real-world test cases from different problem domains by users of the SUMO Toolbox in several other studies. Because of its efficiency, it has since become the default sequential design strategy for the SUMO Toolbox.

LOLA–Voronoi was designed as a very robust, reliable, and widely applicable sequential design method, able to produce good results with any model type, regardless of the problem at hand. To achieve this, the only information used to guide the sampling process consists of previously evaluated samples and their output values. The robustness of the sequential design method comes at a cost, however. It is rather slow compared to other sequential design methods, mainly due to the expensive preprocessing required to estimate the gradient ($O(n^2)$ in the number of samples). However, this additional cost becomes negligible in a real-life environment in which sample evaluations may take hours or even days, and many heuristics are available to substantially reduce the overhead of LOLA–Voronoi.

In future work, the newly proposed method will be further compared and tested against other sequential design methods on real-world problems. The relative effect of the weights of both components (LOLA and Voronoi) on the accuracy of the models will also be tested, and it will be investigated whether a simulated annealing approach produces better results. Additionally, we will study how the algorithm can be extended to support constrained problems and simulators with multiple outputs.

Acknowledgments. The authors would like to thank Jeroen Croon from the NXP-TSMC Research Centre, Eindhoven, The Netherlands for making the LNA code available.

REFERENCES

[1] F. Aurenhammer, *Voronoi diagrams—A survey of a fundamental geometric data structure*, ACM Comput. Surveys, 23 (1991), pp. 345–405.

[2] D. Busby, C. L. Farmer, and A. Iske, *Hierarchical nonlinear approximation for experimental design and statistical data fitting*, SIAM J. Sci. Comput., 29 (2007), pp. 49–69.

[3] H. Cohn and A. Kumar, *Universally optimal distribution of points on spheres*, J. Amer. Math. Soc., 20 (2007), pp. 99–148.

[4] I. Couckuyt, D. Gorissen, H. Rouhani, E. Laermans, and T. Dhaene, *Evolutionary regression modeling with active learning: An application to rainfall runoff modeling*, in Proceedings of the International Conference on Adaptive and Natural Computing Algorithms, Ljubljana, Slovenia, 2009, pp. 548–558.

[5] H. T. Croft, K. J. Falconer, and R. K. Guy *Unsolved Problems in Geometry*, Springer, Berlin, New York, 1994.

[6] K. Crombecq, I. Couckuyt, D. Gorissen, and T. Dhaene, *Space-filling sequential design strategies for adaptive surrogate modeling*, in Proceedings of the First International Conference on Soft Computing Technology in Civil, Structural and Environmental Engineering, Civil-Comp Press, Stirling, UK, 2009, paper 50.

[7] K. Crombecq, D. Gorissen, L. De Tommasi, and T. Dhaene, *A novel sequential design strategy for global surrogate modeling*, in Proceedings of the 2009 Winter Simulation Conference, IEEE Press, Piscataway, NJ, 2009, pp. 731–742.

[8] K. T. Fang, *Experimental design by uniform distribution*, Acta Math. Appl. Sin., 3 (1980), pp. 363–372.

[9] A. Farhang-Mehr and S. Azarm, *Bayesian meta-modeling of engineering design simulations: A sequential approach with adaptation to irregularities in the response behavior*, Internat. J. Numer. Methods Engrg., 62 (2005), pp. 2104–2126.

[10] A. Forrester, A. Sobester, and A. Keane, *Engineering Design Via Surrogate Modeling: A Practical Guide*, John Wiley, New York, 2008.

[11] M. Fu, *Stochastic Gradient Estimation*, preprint version of Chapter 19, "Gradient Estimation," in Handbooks in Operations Research and Management Science: Simulation, S. G. Henderson and B. L. Nelson, eds., Elsevier, Amsterdam, 2005, pp. 575-616.

[12] M. C. Fu and S. D. Hill, *Optimization of discrete event systems via simultaneous perturbation stochastic approximation*, IEEE Trans., 29 (1997), pp. 233–243.

[13] J. Gehrke, V. Ganti, R. Ramakrishnan, and W. Y. Loh, *Boat—Optimistic decision tree construction*, SIGMOD Record, 28 (1999), pp. 169–180.

[14] D. Gorissen, K. Crombecq, I. Couckuyt, and T. Dhaene, *Automatic approximation of expensive functions with active learning*, in Foundations of Computational Intelligence, Learning and Approximation, Vol. 1, Springer-Verlag, Berlin, New York, 2008, pp. 35-62.

[15] D. Gorissen, K. Crombecq, I. Couckuyt, T. Dhaene, and P. Demeester, *A surrogate modeling and adaptive sampling toolbox for computer based design*, J. Mach. Learning Res., 11 (2010), pp. 2051–2055.

[16] D. Gorissen, K. Crombecq, W. Hendrickx, and T. Dhaene, *Adaptive distributed metamodeling*, in Proceedings of the 7th International Conference on High Performance Computing for Computational Science (VECPAR 2006), Lecture Notes in Comput. Sci. 4395, Springer-Verlag, Berlin, 2007, pp. 579–588.

[17] D. Gorissen, L. De Tommasi, K. Crombecq, and T. Dhaene, *Sequential modeling of a low noise amplifier with neural networks and active learning*, Neural Comput. Appl., 18 (2009), pp. 485–494.

[18] D. Gorissen, L. De Tommasi, J. Croon, and T. Dhaene, *Automatic model type selection with heterogeneous evolution: An application to rf circuit block modeling*, in Proceedings of the IEEE World Congress on Computational Intelligence (WCCI 2008), IEEE Press, Piscataway, NJ, 2008, pp. 989–996.

[19] D. Gorissen, F. De Turck, and T. Dhaene, *Evolutionary model type selection for global surrogate modeling*, J. Mach. Learning Res., 10 (2009), pp. 2039–2078.

[20] R. Gramacy and H. K. H. Lee, *Adaptive Design of Supercomputer Experiments*, Technical report, Department of Applied Mathematics and Statistics, University of California, Santa Cruz, CA, 2006.

[21] T. Hachisuka, W. Jarosz, R. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. Wann Jensen, *Multidimensional adaptive sampling and reconstruction for ray tracing*, ACM Trans. Graphics, 27 (2008), pp. 1–10.

[22] W. Hendrickx and T. Dhaene, *Sequential design and rational metamodeling*, in Proceedings of the 2005 Winter Simulation Conference, N. Steiger and M. E. Kuhl, eds., IEEE Press, Piscatoway, NJ, 2005, pp. 290–298.

[23] A. A. Jamshidi and M. J. Kirby, *Towards a black box algorithm for nonlinear function approximation over high-dimensional domains*, SIAM J. Sci. Comput., 29 (2007), pp. 941–963.

[24] R. Jin, W. Chen, and A. Sudjianto, *On sequential sampling for global metamodeling in engineering design*, in Proceedings of DETC'02: ASME 2002 Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Montreal, Canada, 2002, pp. 539–548.

[25] J. P. C. Kleijnen and W. C. M. van Beer, *Application-driven sequential designs for simulation experiments: Kriging metamodeling*, J. Oper. Res. Soc., 55 (2004), pp. 876–883.

[26] J. Knowles and H. Nakayama, *Meta-modeling in multiobjective optimization*, in Multiobjective Optimization: Interactive and Evolutionary Approaches, Springer-Verlag, Berlin, New York, 2008, pp. 245–284.

[27] T. H. Lee, *The Design of CMOS Radio-Frequency Integrated Circuits*, 2nd ed., Cambridge University Press, Cambridge, UK, 2004.

[28] R. Lehmensiek and P. Meyer, *Creating accurate multivariate rational interpolation models of microwave circuits by using efficient adaptive sampling to minimize the number of computational electromagnetic analyses*, IEEE Trans. Microwave Theory Tech., 49 (2001), pp. 1419–1430.

[29] R. Lehmensiek, P. Meyer, and M. Müller, *Adaptive sampling applied to multivariate, multiple output rational interpolation models with application to microwave circuits*, Internat. J. RF Microwave Computer-Aided Engrg., 12 (2002), pp. 332–340.

[30] Y. Lin, *An Efficient Robust Concept Exploration Method and Sequential Exploratory Experimental Design*, Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, 2004.

[31] D. J. C. MacKay, *Bayesian model comparison and backprop nets*, in Advances in Neural Information Processing Systems 4, Morgan Kaufmann, San Francisco, 1992, pp. 839–846.

[32] I. G. Osio and C. H. Amon, *An engineering design methodology with multistage Bayesian surrogates and optimal sampling*, Res. Engrg. Design, 8 (1996), pp. 189–206.

[33] F. J. Provost, D. Jensen, and T. Oates, *Efficient progressive sampling*, in Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, 1999, pp. 23–32.

[34] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn, *Design and analysis of computer experiments*, Statist. Sci., 4 (1989), pp. 409–435.

[35] E. B. Saff and A. B. J. Kuijlaars, *Distributing many points on a sphere*, Math. Intell., 19 (1997), pp. 5–11.

[36] M. J. Sasena, *Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations*, Ph.D. thesis, University of Michigan, Ann Arbor, MI, 2002.

[37] J. Shekel, *Test functions for multimodal search techniques*, in Proceedings of the Fifth Annual Princeton Conference on Information Science and Systems, 1971, pp. 354–359.

[38] T. W. Simpson, D. K. J. Lin, and W. Chen, *Sampling strategies for computer experiments: Design and analysis*, Internat. J. Reliability Appl., 2 (2001), pp. 209–240.

[39] T. W. Simpson, J. Peplinski, P. N. Koch, and J. K. Allen, *Metamodels for computer-based engineering design: Survey and recommendations*, Engrg. Comput., 17 (2001), pp. 129–150.

[40] M. Sugiyama, *Active learning in approximately linear regression based on conditional expectation of generalization error*, J. Mach. Learning Res., 7 (2006), pp. 141–166.

[41] C. J. Turner, R. H. Crawford, and M. I. Campbell, *Multidimensional sequential sampling for nurbs-based metamodel development*, Engrg. Comput., 23 (2007), pp. 155–174.

[42] J. R. Wieland and B. W. Schmeiser, *Stochastic gradient estimation using a single design point*, in Proceedings of the 2006 Winter Simulation Conference, IEEE Press, Piscataway, NJ, 2006, pp. 390–397.