Generating Sequential Space-Filling Designs Using Genetic Algorithms and Monte Carlo Methods

Karel Crombecq[1] and Tom Dhaene[2]

[1] University of Antwerp, 2020 Antwerp, Belgium
Karel.Crombecq@ua.ac.be
[2] Ghent University - IBBT, 9050 Ghent, Belgium

Abstract. In this paper, the authors compare a Monte Carlo method and an optimization-based approach using genetic algorithms for sequentially generating space-filling experimental designs. It is shown that Monte Carlo methods perform better than genetic algorithms for this specific problem.

Keywords: surrogate modelling, active learning, sequential design, Monte Carlo, genetic algorithm, space-filling.

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 behaviour 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 hours to compute [2]. 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.

The goal of *global* surrogate modelling is to find a function that mimics the original system, but can be computed 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.

It is clear that the choice of the data points (or samples) is of paramount importance to the success of the surrogate modelling task. Intuitively, the data points must be spread out in such a way as to convey a maximum amount of

K. Deb et al. (Eds.): SEAL 2010, LNCS 6457, pp. 80–84, 2010.

information about the behaviour of simulator. This is a non-trivial challenge, since little or nothing is known about this (black-box) simulator in advance. From now on, we will refer to the entire set of samples that were selected for evaluation as the experimental design.

Sequential design (which is also known as adaptive sampling [7] or active learning [10]) methods generate an experimental design by selecting samples one by one, allowing for an integrated approach with the surrogate modelling task. After each simulation, new models are built, and the accuracy of these models is estimated. If the target accuracy is reached, the algorithm is halted. Otherwise, another sample is selected and the process starts all over again.

In this paper, we study the sequential generation of space-filling designs. Space-filling designs attempt to spread out the samples as evenly as possible, in order to get as much information about the entire design space as possible. We will investigate why it is difficult to sequentially generate good space-filling designs, and we will compare two approaches to tackling this problem: optimization using genetic algorithms, and Monte Carlo methods.

2 Space-Filling Experimental Design Criteria

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 $\mathbf{p}_i = \left(p_i^1, p_i^2, \ldots, p_i^d\right)$ in the (hyper)cube $[-1, 1]^d$. This experimental design P will be constructed by selecting samples one by one, without knowing the total number of samples n at any point during the construction process. In order to evaluate the space-filling qualities of the final design, we consider two criteria.

First and foremost, the generated design should be space-filling. Intuitively, a space-filling design is an experimental design in which the points are spread out evenly over the design space. However, there are several ways do define this property mathematically. Over the course of the years, many different space-filling criteria have been proposed. Depending on the criterion, the optimal design P will look differently. A popular and intuitive choice is the maximin criterion or *intersite distance* [1,5,6,8,9,12]. The intersite distance of an experimental design P is the smallest distance between two points in the design, and is defined as follows:

$$\text{idist}(P) = \min_{\mathbf{p_i}, \mathbf{p_j} \in P} \sqrt{\sum_{k=1}^{d} \left\| p_i^k - p_j^k \right\|^2} \tag{1}$$

Secondly, a good space-filling design should also have good projective properties. This is also called the non-collapsing property by some authors [1]. An experimental design P has good projective properties if, for every point $\mathbf{p}_i$, each value p_i^j is strictly unique, and as different from the other values as possible. This property also means that, when the experimental design is projected from d-dimensional space to $(d-1)$-dimensional space along one of the axes, no two points are ever projected onto the same location. The quality of a design in terms

of its projective properties can be defined as the minimum *projected distance* of points from each other:

$$\begin{aligned} \text{pdist}(P) &= min_{\mathbf{p_i},\mathbf{p_j}\in P} \min_{1\leq k\leq d} \left|p_i^k - p_j^k\right| \\ &= min_{\mathbf{p_i},\mathbf{p_j}\in P} \left\|\mathbf{p_i} - \mathbf{p_j}\right\|_{-\infty} \end{aligned} \tag{2}$$

3 Results

Generating an experimental design that satisfies the two criteria mentioned in the previous section is a multi-objective optimization problem. Starting with two initial points (for example, opposing corner points), a new point is selected by finding a location in the design space that maximizes both the intersite and projective distance. Many different methods have been proposed to solve such multi-objective optimization problems efficiently. The simplest approach is to combine the different objectives in a single aggregate objective function. This solution is only acceptable 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 intersite and projective distance, this is indeed the case.

The final objective function, which scores a new candidate point $\mathbf{p}$ when it is added to an existing design P, is defined as:

$$\text{dist}(P,\mathbf{p}) = \frac{(n+1)^{\frac{1}{d}-1}}{2} \text{min}_{\mathbf{p_i}\in P} \sqrt{\sum_{k=1}^{d} \left|p_i^k - p^k\right|^2} + \frac{n+1}{2} \text{min}_{\mathbf{p_i}\in P} \left\|\mathbf{p_i} - \mathbf{p}\right\|_{-\infty} \tag{3}$$

In this paper, we will compare two approaches to finding the best location for the next point at each iteration. The first one is a Monte Carlo method. In this method a large number of uniformly distributed random points is generated, and for each candidate $\mathbf{p}$, the objective function $\text{dist}(P,\mathbf{p})$ is calculated and the best candidate is selected as the new point to be added to P. In the second method, the genetic algorithm toolbox from Matlab will be used to optimize the objective function and find the best next candidate. Both methods will be compared for different settings, and the final designs be evaluated on the `idist` and `pdist` criteria to compare both approaches.

At each iteration, the Monte Carlo method will generate kn random points, where n is the number of samples evaluated thus far, and k is an algorithm parameter. It is expected that, for larger k, the quality of the design will improve. In this study, the following values for k were considered: $50, 250, 2000, 10000, 50000$.

For the genetic algorithm, the implementation from the Matlab Genetic Algorithm and Direct Search Toolbox (version 3.0) was used. Most of the options were kept at their default values, but some were changed in order to improve the performance. The default mutation function (which offsets each input by

a value drawn from a gaussuan distribution) wasn't usable, because it did not respect the boundary constraints (each input must lie in $[-1, 1]$). It was changed to a mutation function that changes each input with a chance of 0.01 to a random values in the $[-1, 1]$ interval. Preliminary results have shown that playing with the crossover/mutation fraction settings, changing the elite behaviour etc does not affect the outcome much, so these settings were kept at their default values. This experiment was repeated for different numbers of generations: $50, 100, 250, 1000, 2000$.

All these experiments were carried out using the SUMO Toolbox research platform [3,4]. This freely available Matlab toolbox, designed for adaptive surrogate modelling and sampling, has excellent extensibility, making it possible for the user to add, customize and replace any component of the sampling and modelling process. Because of this, SUMO was the ideal choice for conducting this experiment[1].

In order to compare the two methods, an experimental design of 144 points was generated in a $2D$ input space, and the intersite and projective distance of the final design generated by both methods were compared. Because both the Monte Carlo and genetic algorithms use random numbers, each experiment was repeated 10 times to get a good average of the performance of the methods. This demonstrated a clear trend: even the Monte Carlo method with $k = 50$ produced better results than the genetic algorithm with 2000 generations. And even though the final designs generated by the genetic algorithm are considerably worse than the ones generated by the Monte Carlo method, the genetic algorithm requires much more time to generate them. The genetic algorithm with 2000 generations takes 3 times longer than the Monte Carlo method with $k = 50$, but still produces worse results.

It is also noticeable that the difference between 50 generations and 2000 generations is smaller than the difference between $k = 50$ and $k = 50000$, while the difference in elapsed time is larger for the genetic algorithm. This indicates that the rate at which the genetic algorithm improves is actually lower than the improvement rate for the Monte Carlo method. So no matter how many generations are computed, there will always be a Monte Carlo alternative that requires less time to get the same result.

4 Conclusion

This study shows that, when sequentially generating space-filling experimental designs, Monte Carlo methods are prefered above genetic algorithms or other optimization methods. This can be explained by the extremely complex and multimodal optimization surface obtained by adding the intersite and projective distance. It is possible that other optimization methods might perform better than the genetic algorithm implementation from the Matlab toolbox used in this study. However, the authors find it unlikely that any optimization method

[1] The SUMO Toolbox v7.0 can be downloaded from `http://www.sumo.intec.ugent.be`

will do better than the Monte Carlo approach, considering the nature of the optimization surface.

Preliminary experiments have shown that the results published in this paper also hold in higher dimensions, and for different criteria, such as the ϕ_p criterion proposed by [8]. In subsequent publications, these preliminary results will be examined and expanded upon.

The ultimate goal of this experiment is to develop highly efficient sequential space-filling algorithms that can compete with proven and popular one-shot experimental design techniques such as the optimized Latin hypercube [1,11]. These methods will use Monte Carlo methods as the optimization method of choice, as opposed to global optimization methods. Finally, hybrid methods will also be investigated. Hybrid methods use Monte Carlo to find promising locations, and then perform a local optimization to further improve the initial result.

References

1. van Dam, E.R., Husslage, B., den Hertog, D., Melissen, H.: Maximin latin hypercube design in two dimensions. Operations Research 55(1), 158–169 (2007)
2. Gorissen, D., Crombecq, K., Hendrickx, W., Dhaene, T.: Adaptive distributed metamodeling. In: Daydé, M., Palma, J.M.L.M., Coutinho, Á.L.G.A., Pacitti, E., Lopes, J.C. (eds.) VECPAR 2006. LNCS, vol. 4395, pp. 579–588. Springer, Heidelberg (2007)
3. Gorissen, D., Tommasi, L.D., Crombecq, K., Dhaene, T.: Sequential modeling of a low noise amplifier with neural networks and active learning. Neural Computation & Applications 18(5), 485–494 (2009)
4. Gorissen, D., Turck, F.D., Dhaene, T.: Evolutionary model type selection for global surrogate modeling. Journal of Machine Learning Research 10(1), 2039–2078 (2009)
5. Johnson, M., Moore, L., Ylvisaker, D.: Minimax and maximin distance designs. Journal of Statistical Planning and Inference 26, 131–148 (1990)
6. Joseph, V.R., Hung, Y.: Orthogonal-maximin latin hypercube designs. Statistica Sinica 18, 171–186 (2008)
7. Lehmensiek, R., Meyer, P., Müller, M.: 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(4), 332–340 (2002)
8. Morris, M.D., Mitchell, T.J.: Exploratory designs for computer experiments. Journal of Statistical Planning and Inference 43, 381–402 (1995)
9. Rennen, G., Husslage, B., Dam, E.V., Hertog, D.D.: Nested maximin latin hypercube designs. Tech. Rep., Tilburg University (2009)
10. Sugiyama, M.: Active learning in approximately linear regression based on conditional expectation of generalization error. Journal of Machine Learning Research 7, 141–166 (2006)
11. Viana, F.A.C., Venter, G., Balabanov, V.: An algorithm for fast optimal latin hypercube design of experiments. International Journal for Numerical Methods in Engineering (2009)
12. Ye, K.Q., Li, W., Sidjianto, A.: Algorithmic construction of optimal symmetric latin hypercube designs. Journal of Statistical Planning and Inference 90(1), 145–159 (2000)