Computational Statistics and Data Analysis 144 (2020) 106844

Contents lists available at ScienceDirect

Computational Statistics and Data Analysis

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

A comparison of general-purpose optimization algorithms for finding optimal approximate experimental designs

Ricardo García-Ródenas [a,*], José Carlos García-García [a], Jesús López-Fidalgo [b], José Ángel Martín-Baos [a], Weng Kee Wong [c]

[a] Departamento de Matemáticas, Escuela Superior de Informática, Universidad de Castilla la Mancha, 13071–Ciudad Real, Spain
[b] Universidad de Navarra, ICS, Campus Universitario, 31080–Pamplona, Spain
[c] Department of Biostatistics, University of California, Los Angeles, USA

ARTICLE INFO

Article history:
Received 9 October 2018
Received in revised form 1 August 2019
Accepted 21 September 2019
Available online 10 October 2019

Keywords:
Approximate design
Efficiency
Equivalence theorem
Information matrix
Metaheuristics
Optimality criteria

ABSTRACT

Several common general purpose optimization algorithms are compared for finding *A*- and *D*-optimal designs for different types of statistical models of varying complexity, including high dimensional models with five and more factors. The algorithms of interest include exact methods, such as the interior point method, the Nelder–Mead method, the active set method, the sequential quadratic programming, and metaheuristic algorithms, such as particle swarm optimization, simulated annealing and genetic algorithms. Several simulations are performed, which provide general recommendations on the utility and performance of each method, including hybridized versions of metaheuristic algorithms for finding optimal experimental designs. A key result is that general-purpose optimization algorithms, both exact methods and metaheuristic algorithms, perform well for finding optimal approximate experimental designs.

1. Introduction

Optimal design of experiments as a subfield of statistics has a long history and dates back to the early 1900s (Smith, 1918). Historically, the focus was on polynomial models and optimal designs were found from theoretical considerations. Beyond homoscedastic polynomial models, esoteric theory from mathematical subfields is frequently required to find theoretical optimal designs. Because of the mathematical complexity, the bulk of the theoretically developed optimal designs invariably have one or two factors or models are assumed to be additive when there are multiple factors. Some closed-form designs can be found in Johnson and Nachtsheim (1983), López-Fidalgo and Wong (2002) or Amo-Salas et al. (2016), to mention a few.

High dimensional models are of increasing interest because they reflect studies more realistically and aided by increase in computer power. For definiteness, we call models high dimensional if it has 5 or more factors in the model. The problem to find an analytical description of the optimal design for a high dimensional model becomes more challenging because they have many variables to optimize. The implications are that relying solely on theory to find optimal designs can be limiting in a number of ways. First, the theory developed for finding an optimal design or a class of optimal designs is frequently very dependent on the model assumptions. For example, if the mean function is slightly changed, the mathematical derivation of the optimal design becomes invalid and may not provide clues for modifications or extensions

* Corresponding author.
E-mail addresses: Ricardo.Garcia@uclm.es (R. García-Ródenas), JoseCarlos.Garcia@uclm.es (J.C. García-García), fidalgo@unav.es (J. López-Fidalgo), JoseAngel.Martin@uclm.es (J.Á. Martín-Baos), wkwong@ucla.edu (W.K. Wong).

https://doi.org/10.1016/j.csda.2019.106844
0167-9473/

to tackle the new optimization problem. Second, some assumptions required in the proof may be unrealistic and are merely imposed to arrive at an analytical description of the optimal design. Third, the theory may apply to a particular setup only. As an example, when the design criterion is convex, the elegant equivalence theorem verifies whether a design is optimal among all designs on the design space but it is not applicable when one is only interested to confirm whether a design is optimal within a smaller non-convex class.

Algorithms are a practical way to find optimal designs for a specific criterion and model. They are appealing because the practitioner can run an algorithm and use the generated design quickly without having to resort to theory to derive the design. There are several types of algorithms. Some are specific for a particular problem and others are general purpose optimization algorithms. Some are exact with mathematical proofs of convergence and others are based on heuristics. Some of the challenges with an algorithmic approach are that some algorithms may not work when the criterion is non-differentiable or non-convex. This means that algorithms that utilize gradient-based methods or convexity properties of the problems may no longer apply. Similarly, algorithms that work well for small dimensional problems may not work well when a model incorporates more factors in the model because the number of design variables that need to be optimized can increase quickly. For example, a model with 5 factors and all pairwise interaction terms requires at least 16 design points to estimate all parameters. This implies we have to optimize at least 95 design variables because we do not know how many points the optimal design has.

In the last few decades, a special class of heuristic algorithms called nature-inspired metaheuristics algorithms has emerged as particularly powerful and effective for tackling various types of optimization problems. This class does not require assumptions to perform well, is flexible and easy to implement and use. Many of these are not widely used in mainstream statistics even though they are already widely used in engineering and computer science. A motivation for this paper is to describe some of these newer types of algorithms and use simulations to compare their performances with traditional algorithms used in statistics for finding optimal designs. We focus on design problems but many of these heuristic algorithms are general purpose and can be used for solving other types of design problems and non-design statistical problems. For example, nature-inspired algorithms were used in Chen et al. (2013) to optimize Latin Hypercube designs and in Leatherman et al. (2014) to construct optimal computer experiments. Such algorithms have also been used to provide optimal estimates for investigating efficacy of dual lung cancer screening by chest X-ray and sputum cytology (Kim et al., 2012) and estimating parameters in a nonlinear mixed PK/PD model for a pharmaceutical application (Kim and Li, 2011). These algorithms are versatile and each has their own appeal; for example, Genetic Algorithms (Lin et al., 2015) or robust optimization techniques (Mak and Wu, 2019) are excellent algorithms for solving optimization problems over a discrete search space.

The main purpose of this paper is to conduct a broad evaluation of the relative usefulness of deterministic algorithms and nature-inspired metaheuristic algorithms and their hybrids for finding various optimal experimental designs. We also compare their relative performances, in terms of speed and ability to find an optimal design for linear and nonlinear models with varying complexity of the optimization problem. Previous work typically compared algorithms of the same type. For example, Cook and Nachtsheim (1980) compared deterministic algorithms, and Hassan et al. (2005) compared among metaheuristic algorithms. The scope of our work is therefore broader and more ambitious in that we compare performances across different types of algorithms and ascertain whether nature-inspired metaheuristic algorithms or their hybrids tend to outperform deterministic algorithms, on average, for searching optimal designs.

The main contributions in this paper are:

1. We provide a brief review of algorithms for finding optimal designs, including nature-inspired metaheuristic algorithms.
2. We perform an empirical analysis of the convergence of heuristic methods, such as, Particle Swarm Optimization (PSO), Simulated Annealing (SA) and Genetic Algorithms (GA) for finding *D*-optimal and *A*-optimal designs.
3. We hybridize each of the heuristic algorithms with the Nelder–Mead algorithm and evaluate whether the hybridized versions are more effective for finding the optimal designs.
4. We implement three exact optimization methods: Sequential Quadratic Programming (SQP), the Active-Set (AS) method and the Interior Point Method (IPM) and compare their ability for finding optimal designs in high dimensional problems with many variables to optimize.

In Section 2, we review the statistical background, different types of designs and various design criteria. Section 3 concerns optimizing a convex functional of the Fisher information matrix, reviews equivalence theorems and discusses a theoretical lower bound for the efficiency for a design without knowing the optimum. This tool is useful because a search algorithm can use it as a stopping rule to terminate the search as soon as it finds a design with the specified minimum efficiency requirement. The section ends with a reformulation of the design problem that is more appropriate for direct applications of many algorithms. Section 4 discusses different types of algorithms for optimization and they include exact methods and nature-inspired metaheuristic algorithms, and concludes with a subsection on how the latter may be hybridized for better performance. Section 5 describes the scope of our simulation study using different types of statistical models, optimality criteria and algorithms. We conclude the section with results from the simulation study. Section 6 presents a summary of our results and general recommendations concerning the choice of algorithms for generating optimal experimental designs.

2. Approximate design, information matrix and optimality criteria

Our statistical models have the form

$$Y = \eta(\mathbf{x}, \theta) + \varepsilon, \quad \mathbf{x} \in X, \tag{1}$$

where Y is a univariate response and $\mathbf{x}$ is a vector of design factors defined on a given compact design space X. The mean response function is η and is assumed known apart from an unknown vector of parameters θ of dimension k. Observations may be correlated and, if so, we assume the covariance structure is known and depends only on θ. The typical goal in the study is to estimate parameters in the model or to estimate a meaningful function of the model parameters. For instance, in a dose response study with a curvilinear mean response, there may be interest in estimating the dose at which the turning point occurs. In this case, one finds a design that minimizes the asymptotic variance of the estimated dose of the turning point.

Throughout, we assume we have a statistical model, design criterion and a given number, n, of observations for the study. The design problem is to optimize the number of design points, their locations within a user-defined design space and the number of replicates to be taken at these design points. If the optimization is over all such designs in the design space, the resulting designs are called exact optimal designs. There is no unified framework for finding them and confirming whether a design is an exact optimal design; see, for example, Esteban-Bravo et al. (2016), who applied algorithms to find exact optimal designs with constraints.

Given a statistical model with normally distributed and independent errors with means 0 and constant variances, the normalized Fisher information matrix of an approximate design ξ for model (1) is proportional to

$$\mathbf{I}(\xi, \theta) = \sum_{i=1}^{m} \omega_i \mathbf{I}(\mathbf{x}_i, \theta), \tag{2}$$

where

$$\mathbf{I}(\mathbf{x}_i, \theta) = \frac{\partial \eta(\mathbf{x}_i, \theta)}{\partial \theta} \frac{\partial \eta(\mathbf{x}_i, \theta)}{\partial \theta^T}$$

is the information matrix from an observation at the point $\mathbf{x}_i$. The inverse of the information matrix is asymptotically proportional to the covariance matrix of the estimators of the parameters of the model. Clearly, for one covariate the information matrix is nonsingular if there are as many distinct design points as the number of parameters in the model, i.e. $k \leq m$. It is worth mentioning that this condition is not required for experiments with multivariate response (see e.g. Yang et al., 2017). This matrix is then maximized in some ways if one wishes to estimate the model parameters accurately by an appropriate choice of the number of design points, the locations of the design points and the weights at each of these points. Kiefer (1974) proposed a class of design criteria that seems quite adequate for many practical problems in which the sample size is large. For a model with k parameters, the class is indexed by a single parameter p with different values of p representing various design criteria:

$$\Phi_p(\mathbf{I}(\xi, \theta)) = \left(\frac{1}{k} Tr\left(\mathbf{I}^{-p}(\xi, \theta) \right) \right)^{\frac{1}{p}}, 0 \leq p \leq \infty. \tag{3}$$

When $p = 1$, we have A-optimality and the optimal design minimizes the average of the variances of the parameter estimates. When p is very large, $\lim_{p\to\infty} \Phi_p(\mathbf{I}(\xi, \theta)) = \lambda_{max}$, the maximum eigenvalue of the information matrix, and we obtain E-optimality. This criterion minimizes the longest axis of the confidence ellipsoid of the parameters. When p is close to 0, $\lim_{p\to 0} \Phi_p(\mathbf{I}(\xi, \theta)) = |\mathbf{I}(\xi, \theta)|$ and we have D-optimality. This criterion provides the design producing the minimum volume of the confidence ellipsoid. The D-, A- and E-optimality criteria make the information matrix large in various ways and result in different design criteria for estimating model parameters. D-optimality is by far the most popular in practice.

Some criteria, in particular A-optimality, have been criticized by researchers because there is a scale-dependence on the parameters in nonlinear models (see e.g. Stallings and Morgan, 2015). Thus, when appropriate a weighted criterion must be used to standardize the corresponding variances and covariances of the estimators of the parameters. In our examples, there are some differences of magnitude in the nominal values of the parameters, but not particularly large, so this problem is not a big issue here. In any case the main goal of the paper is to compare the efficiency of the various algorithms for computing optimal designs. For more practical computations, some weighted criteria should be used.

For nonlinear models, the design criterion contains the unknown parameter θ which we want to estimate. The simplest approach is to assume nominal values for θ are available from previous or similar studies; these values replace the unknown parameters and so the criterion can be optimized by choice of the number and locations of the design points and the number of replications required at each design point. Such optimal designs are termed locally optimal (Chernoff, 1953) because they depend on the nominal value of θ. Clearly, locally optimal designs depend on the nominal values θ and can be sensitive to its value. This means a small mis-specification of the nominal value of θ can result in a noticeable drop in efficiency of the design. Minimax or Bayesian approaches are alternative ways to overcome such issues but are outside the scope of this paper. Throughout, we focus on locally optimal designs.

3. Efficiencies, equivalence theorems, efficiency lower bounds and a reformulation

This section discusses basic tools in optimal design theory. The first subsection describes the concept of design efficiency to measure the worth of a design, the second subsection reviews how a design can be verified to be optimum among all designs when the design criterion is convex (or concave) and the third subsection provides a tool for evaluating how close a design is to the optimum without knowing the optimum in terms of efficiency. The last subsection describes a reformulation of the design problem that is generally helpful to consider before applying an algorithm. For expository purposes in this section, we assume the model is linear since extension to nonlinear models is straightforward; the information matrix below therefore does not contain θ.

3.1. Design efficiencies

We recall that a function Φ is positive homogeneous if, for every positive δ, $\Phi(\delta\xi) = \Phi(\xi)/\delta$. Throughout, we denote our design criterion by $\Phi(\xi)$, sometimes $\Phi(\mathbf{I}(\xi))$, and assume it is both positive homogeneous and convex over the design space. We measure the worth of a design ξ by its Φ-efficiency $\text{eff}_\Phi(\xi) = \Phi(\xi^\star)/\Phi(\xi)$, where $\xi^\star$ is a Φ-optimal design. The efficiency is always between 0 and 1 and it is frequently multiplied by 100 and reported as a percentage. The higher the efficiency, the closer the design ξ is to the optimum. The positive homogeneity of the criterion provides a practical interpretation, i.e. if the efficiency of ξ is 50%, then the design ξ needs to be replicated twice to perform as well as the optimal design $\xi^\star$. To ensure this simple interpretation, the D-efficiency of a design ξ is appropriately modified to be $(|\mathbf{I}(\xi)|/|\mathbf{I}(\xi_D)|)^{1/k}$, where ξ_D is the D-optimal design and k is the dimension of θ.

3.2. Equivalence theorems

When the design criterion is a convex or concave function of the information matrix, one can directly use directional derivative considerations to obtain an equivalence theorem and verify the optimality of a design. As an example, consider D-optimality, where we want to find a design that minimizes the convex functional $\Phi_D(\xi) = -\log|\mathbf{I}(\xi)|$. Let $s(\mathbf{x}, \xi^*)$ be the directional derivative of Φ at a given design ξ^* in the direction toward a degenerate design at the point $\mathbf{x}$. Using convex analysis argument, one arrives at the following equivalence theorem for a homoscedastic linear model: a design ξ_D is D-optimal for all $\mathbf{x}$ in the design space, if and only if

$$s(\mathbf{x}, \xi_D) = f(\mathbf{x})^T \mathbf{I}^{-1}(\xi_D) f(\mathbf{x}) - k \leq 0,$$

with equality at the support points of ξ_D. We note that each convex criterion has its own equivalence theorem; see details in Berger and Wong (2009), Fedorov (1972), Silvey (1980) and Atwood (1976).

3.3. Efficiency lower bounds

The equivalence theorems are also helpful in that they provide a lower bound of the efficiency of any design without requiring the optimum to be known. To this end, we first solve the optimization problem

$$\varepsilon_0 = \max_{\mathbf{x}\in X} s(\mathbf{x}, \xi). \tag{4}$$

Following Pazman (1986), an efficiency lower bound for a convex criterion Φ is:

$$\text{eff}_\Phi(\xi) \geq 1 - \frac{\varepsilon_0}{\Phi(\mathbf{I}(\xi))}.$$

Given a design ξ, many algorithms solve (4) by discretizing the design space. The reformulation of the design problem allows the use of general purpose algorithms for searching over the whole continuous design space to find a solution close to the optimum or the optimum itself. When the search space has a large dimension, discretizing the search space is problematic because it can take a long time to generate and to evaluate a fine grid (see Table 10). For this reason, we do not recommend using algorithms that require the search space to be discretized to solve a high dimensional optimization problem, such as in (4).

3.4. Reformulation of the optimization problem

The common algorithms for finding optimal experimental designs work by adding or deleting points iteratively from the current design. A new point is added to the current design by using a convex combination of the single point design and the current design. The single point is specially selected and the weights used in the convex combination can be chosen to achieve monotonic convergence or the fastest convergence. Some algorithms require the search space for the support points to be discretized and others do not. Caratheodory's theorem gives an upper bound on the number of support points an optimal design needs; this upper bound is $k(k+1)/2+1$, where k is the number of parameters in the mean function (Fedorov, 1972). For strictly decreasing criteria, such as D- and A-optimality, this upper bound can be decreased by one.

Suppose an optimal approximate design ξ^* is supported at $\mathbf{x}_1, \ldots, \mathbf{x}_m$ and let ω be the vector of weights at the corresponding points. We represent ξ by $\mathbf{z} = (\omega^T, \mathbf{x}_1^T, \ldots, \mathbf{x}_m^T)^T$ and optimize the variables in the vector subject to constraints

$$\mathbf{l} \leq \mathbf{x}_i \leq \mathbf{u},\ i = 1, \ldots, m, \tag{5}$$

where $\mathbf{l}$ and $\mathbf{u}$ are user-defined vectors of lower and upper bounds for the different factors in the model. Putting them together as $\mathbf{L} = (\mathbf{0}^T, \mathbf{l}^T, \ldots, \mathbf{l}^T)^T$ and $\mathbf{U} = (\mathbf{1}^T, \mathbf{u}^T, \ldots, \mathbf{u}^T)^T$, the constraints in our optimization problem are

$$\mathbf{L} \leq \mathbf{z} \leq \mathbf{U}, \tag{6}$$

with the additional constraint that the weights sum to one. The design problem we wish to solve becomes

$$\begin{array}{ll} \text{Minimize} & \widehat{\Phi}(\mathbf{z}) \\ \text{subject to:} & \mathbf{L} \leq \mathbf{z} \leq \mathbf{U} \\ & \mathbf{1}^T\omega = 1, \end{array} \tag{7}$$

where $\widehat{\Phi}$ is Φ in this new context. The above formulation is versatile because most algorithms are able to handle a set of linear constraints.

If we have n observations for the study and an exact optimal design is sought, the weight ω_i at each support point is constrained to be a multiple of $\frac{1}{n}$ and subject to the requirement that the total number of observations at all the points sums to n. Mathematically,

$$\omega_i = \frac{q_i}{n}$$
$$\sum_{i=1}^{m} q_i = n$$
$$q_i \in \mathbb{N} \cup \{0\},\ i = 1, \ldots, m.$$

We use the following technique to avoid the integer nature of the variables $q_1, \ldots, q_m$. We consider as optimization variables of the problem (7) a support point for each observation, that is, we find a design with the number of support points equal to the number of experimental units. It follows that the variable $\mathbf{z}$ in the problem (7) is now defined by $\mathbf{z} = (\mathbf{x}_1^T, \ldots, \mathbf{x}_n^T)^T$, and the weights at each support point is $\omega_i = \frac{1}{n}$. The resulting formulation is a continuous optimization problem and avoids use of the integer optimization methods. We also observe that the optimal solution of the problem (7) may be such that $\mathbf{x}_i = \mathbf{x}_j$ for some pairs of observations $i \neq j$. Consequently, general purpose algorithms for continuous nonlinear optimization may be used for both exact and approximate designs.

4. Algorithms

Effective algorithms should be flexible, fast, easy to implement, usable in a variety of contexts and for the purpose intended. They should find the optimal designs in an efficient way in terms of number of evaluations of $\widehat{\Phi}$ and CPU time. We do not expect that there is a single algorithm that performs well for all types of optimal designs but we would like to identify the types of algorithms that seem efficient for solving optimal design problems.

All algorithms require a starting design to initiate the search before they start updating the current design using a procedure that varies from one algorithm to another. They also require a user-specified stopping rule for the algorithm to terminate its search and some popular choices are the maximum number of allowed iterations or successive values of the criterion or efficiencies which do not change by more than a pre-specified amount. Algorithms for generating an optimal design can be deterministic or based on metaheuristics. Mandal et al. (2015) provided a recent overview of algorithms for finding optimal designs.

4.1. Deterministic and exact method algorithms

There are many algorithms proposed for finding optimal designs; some guarantee theoretical convergence to the optimal design and others do not. For example, the classic and well known Fedorov–Wynn types of algorithms can be proved to converge to the optimal design (Fedorov, 1972; Wynn, 1972). After a starting design is selected, at each iteration, the algorithm includes a specially selected point into the current design. The selected point depends on the design criterion; for D-optimality, it is a point that maximizes the variance function of the current design. The new design is formed by taking a convex combination of the current design and the selected point, which is a degenerate design. One common sequence of weights used in the convex combination is $1/i$, $i = 2, 3, \ldots$ The sum of this sequence tends to infinity and the sum of the squared terms is finite. These specifications ensure the sequence converges but not too quickly. More specialized sequences are available to accelerate the convergence or to ensure the largest possible increase or decrease in the criterion value.

Deterministic algorithms for finding optimal designs produce the same design when they are run repeatedly using the same set of input parameters. There is no randomness built into the algorithms and they are generally efficient

because they exhibit linear and super-linear rates of convergence. These algorithms typically have a proof of convergence to the theoretical optimum and examples are the Wynn (1972) and Fedorov (1972) algorithms. Exchange algorithms, or coordinate exchange algorithms fall into this category and are effective for finding exact optimal designs when the model has several discrete factors, see for example, Meyer and Nachtsheim (1995) and the references therein. Advances in such algorithms continue to this day. Huang et al. (2019) is a most recent example that illustrates well the utility of such an approach to find a *D*-optimal exact design for estimating the response surface of a nonlinear model.

Multiplicative algorithms like those described in Yu (2011) also can be shown to generate *D*-optimal designs. These algorithms require the design space to be discretized, implying that only the weights at the user-specified grid points have to be optimized. Points with positive weights become support points of the optimal design. An earlier application of multiplicative algorithms to find marginally and conditionally restricted optimal designs was described in Martín-Martín et al. (2007). A faster version of the multiplicative algorithm called cocktail algorithm was proposed by Yu (2011) for finding *D*-optimal designs.

Our experience is that traditional algorithms like those mentioned above may stall at a local optimum or break down because of the huge computational burden even though it can be proved mathematically that they converge to the optimum. We provide three citations who reported similar experiences with traditional algorithms for finding optimal designs. An early one is Chaloner and Larntz (1989) who found both the Wynn (1972) and Fedorov (1972) algorithms very slow when they tried to find *A*- and *c*-Bayesian optimal designs for the two-parameter logistic model and the prior distributions are vague. They used the general optimization algorithm proposed by Nelder and Mead (1965) and found it to be adequate. Similarly, Broudiscou et al. (1996) claimed that traditional algorithms, such as Fedorov–Wynn types of algorithms or exchange algorithms for finding optimal designs cannot be used to find non-standard designs, such as asymmetrical *D*-optimal designs. They found the algorithms performed poorly and were difficult to handle and so cannot be used. They abandoned them for genetic algorithms instead. Similarly, Royle (2002) reported that the traditional exchange algorithms are not practical for finding large spatial designs when the criterion is computationally expensive to evaluate or the discretized design space is too large. These may be reasons why the bulk of the optimal experimental designs reported in the literature concern a small number of factors. However, there are newer versions of coordinate exchange algorithms that do not require the search space to be discretized. Examples are those implemented in JMP 14 (SAS Institute Inc) and in Huang et al. (2019) where they both use a continuous search space.

Exact methods form another class of optimization tools. An exact optimization procedure ensures an optimal solution is found. They include interior point methods (IPMs), the active set method (ASM) and sequential quadratic programming (SQP) and can also be shown to converge to a local optimum. IPMs calculate the Hessian by a dense quasi-Newton approximation. The use of derivatives allows the algorithm to achieve super-linear convergence rates to stationary points of the Lagrangian function. Each evaluation of the objective function usually requires solving a new optimizing problem. Lu and Pong (2013) used an IPM to find optimal experimental designs. Exact methods require assumptions for them to work well. If these are satisfied, they are known to be powerful for solving various kinds of optimization problems. Sun and Yuan (2006) discussed assumptions for various exact methods with examples. For both the ASM and SQP, a quasi-Newton approximation to the Hessian of the Lagrangian has to be computed; details of these programming tools are available from the support pages at www.SAS.com.

4.2. Nature-inspired metaheuristic algorithms

Heuristic algorithms usually have a stochastic component that helps them to avoid or escape from a local optimum. Heuristic algorithms usually do not guarantee convergence to the global optimum but have advantages over deterministic algorithms. For instance, these methods do not require the user to make assumptions on the criterion to be optimized. Moreover, this criterion does not have to be differentiable. A drawback of heuristic algorithms is that they involve tuning parameters, which can have a real impact on the algorithm performance. Typically, there are many suggestions for choices for tuning parameters which can improve performance considerably, see for example Cuervo et al. (2014) and Lin et al. (2015). The algorithms can also be slow to converge and, when this happens, hybridization strategies (described below) are used to speed them up. Some algorithms can be self adaptive meaning that, with repeated runs over time, they improve their own tuning parameters (Qin et al., 2009).

A recent class of heuristic algorithms that has shown exceptional promise for solving a broad class of optimization problems is the class of nature-inspired metaheuristic algorithms. These algorithms are motivated by observing nature. They are marked by a simple algorithm that is easy to implement yet flexible to adapt to solve many types of optimization problems. Such algorithms are widely used in engineering and computer science research and they are constantly being improved. The more recent ones have not made it to mainstream statistical research. We now briefly review some of them below; Yang (2010) provided a broader review of nature-inspired metaheuristic algorithms, including computer codes.

We now briefly discuss three metaheuristic algorithms, namely, simulated annealing, genetic algorithm and particle swarm optimization because there is already a huge literature on them. In each case, we provide a pseudo code for the heuristic algorithm before we mention some of their hybridizations in the next subsection. We let $\widehat{\Phi}$ denote the criterion value to be optimized.

Table 1
The simulated annealing method.

Step 1.	(*Initialization*). Determine the number of iterations (N) and set the counter t to 1. Choose an initial solution $\mathbf{z}^0$. Initialize the temperature T.
Step 2.	Randomly select $\mathbf{z}' \in V(\mathbf{z}^t)$, where V is a neighborhood of $\mathbf{z}^t$.
Step 3.	If $\widehat{\Phi}_p(\mathbf{z}') \leq \widehat{\Phi}(\mathbf{z}^t)$ then $\mathbf{z}^t = \mathbf{z}'$. Otherwise, set $\mathbf{z}^t = \mathbf{z}'$ with probability $q^t = \exp\left(-\frac{\widehat{\Phi}(\mathbf{z}')-\widehat{\Phi}(\mathbf{z}^t)}{T}\right)$
Step 4.	Decrease the temperature T.
Step 5.	(*Stopping criterion*). If the current iteration is $t = N$, Stop; otherwise set $t = t + 1$ and go to Step 2.

Table 2
A prototype genetic algorithm.

Step 1.	(*Initialization*). Determine the number of iterations (N) and set the counter t to 1. Randomly choose a set of solutions (population of individuals).
Step 2.	Select a set of pairs of parents to recombine.
Step 3.	Mutate the resulting offspring.
Step 4.	Evaluate the new individuals.
Step 5.	Select the new individuals for the next generation.
Step 6.	(*Stopping criterion*). If the current iteration is $t = N$, Stop; otherwise set $t = t + 1$ and go to Step 2.

4.2.1. Simulated annealing

The simulated annealing (SA) procedure was proposed independently by Kirpatrick et al. (1983) and Černý (1985). SA is inspired by the cooling techniques used in metallurgy for obtaining a "well ordered" solid state of minimal energy. SA is a probabilistic method, which means that each time the algorithm runs a different solution may be obtained. The main ingredients of the algorithm are: (i) the definition of the neighborhood, (ii) the probability distribution of the neighborhood and (iii) the cooling rule. Table 1 shows a generic scheme of the SA. To the best of our knowledge, the first to apply SA to find optimal designs for linear regression models was Haines (1987). Most recently, López-García et al. (2014) proposed a variant of SA consisting of using as neighborhood of $\mathbf{z}$ all points $\mathbf{z}'$ which differ from $\mathbf{z}$ in a single component of the vector. When the probability distribution on the neighborhood is uniform, we denote the variant by SA*. SA can be shown to converge to the theoretical optimum if the tuning parameters are appropriate.

4.2.2. Genetic algorithms

The Genetic Algorithm (GA) was introduced by Holland (1975) and Goldberg (1989). This book considers a probabilistic–heuristic type of algorithms integrated in so-called population-based procedures of searching for the global optimum within a set of solutions (*population*). The rules operation on the population are based on Darwinian principles simulating the evolution of individuals. Each iteration of the algorithm corresponds to a generation. A generation is obtained using two main mechanisms. One is the so-called *recombination*, also called *cross-over*, where two or more solutions (the so-called *parents*) are combined to get new descendants (the so-called *offspring*). These new solutions are subject to *mutations* that modify their features. There are also rules to select the individuals among the new and old solutions that will remain in the next generation. Table 2 displays a basic GA pseudo code. Hamada et al. (2001) applied GA to find near-optimal Bayesian experimental designs and Latif and Brunner (2016) used GA to design micro-array experiments.

GA is probably one of the most widely used general purpose algorithms for optimization in statistics. There are many applications of GAs in statistics, including for finding different types of optimal designs; see for example, Broudiscou et al. (1996), Su and Chen (2012), etc.

4.2.3. Particle swarm optimization

Particle swarm optimization (PSO) was introduced by Kennedy and Eberhart (1995). It is a probabilistic population-based method and the rules operating in the population are inspired by the flock behavior of birds or swarms of bees. In this algorithm, the elements are called *particles* and each particle movement is autonomous with a random component and connected to a subset of particles of the population, which is known as the neighborhood of the particle. The *velocity* of a particle is determined by (i) its current velocity; (ii) the experience of the particular particle (*personal best*), i.e. the best obtained position in its whole evolution and (iii) the *best position* found by the particles of its neighborhood. These factors are weighted and randomly modified to determine the resulting velocity of the particle.

Table 3 displays a pseudo code for the basic PSO. The algorithm depends on three parameters: (i) the inertia weight ω, (ii) the cognition acceleration coefficient C_1 and (iii) the social acceleration coefficient C_2. The trajectory of an individual particle would converge contingent upon meeting the following condition (see Sengupta et al., 2018)

$$1 > \omega > \frac{C_1 + C_2}{2} - 1 \geq 0. \tag{8}$$

These parameters have been set $\omega = \frac{1}{2\log(2)}$, $C_1 = 0.5 + \log(2)$ and $C_2 = 0.5 + \log(2)$ in the numerical simulations to assure that each particle converges. This point may not be an optimum and particles may prematurely converge to it. However, one appealing property of PSO is that a wide range of values for the parameters in PSO work well for a wide

Table 3
Pseudocode for particle swarm optimization.

Step 1. (*Initialization*). Fix the number of iterations (N) and set the counter t to 1. Initialize the population of particles with random positions $\{\mathbf{z}_i^0\}$ and velocities $\{\mathbf{v}_i^0\}$. Set $\mathbf{p}_i = \mathbf{z}_i^0$ and $\mathbf{g} = \arg\min_i \widehat{\Phi}(\mathbf{z}_i^0)$

Step 2. For each particle i do

- Update the velocity of the particle i using the following equation

$$\mathbf{v}_i^t = \omega \mathbf{v}_i^{t-1} + C_1\varphi_1(\mathbf{p}_i - \mathbf{z}_i^{t-1}) + C_2\varphi_1(\mathbf{g} - \mathbf{z}_i^{t-1}), \tag{9}$$

where φ_1 and φ_2 are two random numbers uniformly distributed on $[0, 1]$, C_1 and C_2 are two parameters known as acceleration coefficients, and ω is the inertia weight.

- Update the position of particle i using the following equation:

$$\mathbf{z}_i^t = \mathbf{z}_i^{t-1} + \mathbf{v}_i^t. \tag{10}$$

- Evaluate the fitness $\widehat{\Phi}(\mathbf{z}_i^t)$ of $\mathbf{z}_i^t$. If $\widehat{\Phi}(\mathbf{z}_i^t) < \widehat{\Phi}(\mathbf{p}_i)$ then $\mathbf{p}_i = \mathbf{z}_i^t$.
- If $\widehat{\Phi}(\mathbf{z}_i^t) < \widehat{\Phi}(\mathbf{g})$ then $\mathbf{g} = \mathbf{z}_i^t$.

Step 3 (*Stopping criterion*). If the current iteration is $t = N$, Stop; otherwise set $t = t + 1$ and go to Step 2.

spectrum of optimization problems (Yang, 2010). In contrast, SA and GA tend to work well only if the right choice set for the tuning parameters is used, but the choice can be problematic in practice; see, for example, Ingber (1993). Marini and Walczak (2015) provided a tutorial on PSO. PSO has been recently used to find various types of optimal designs, see for example, Qiu et al. (2014), Wong et al. (2015) and more recently, Chen et al. (2017), Lukemire et al. (2019) and Mak and Joseph (2018). The first set of authors provide numerous types of optimal designs found by PSO for different biomedical problems and the second set of authors found optimal designs for several types of mixture models. Chen et al. (2017) found standardized maximin optimal designs for several inhibition models. Lukemire et al. (2019) solved several *D*-optimal design problems involving binary responses using an improved version of PSO.

4.3. Hybridization of metaheuristics

Global optimization methods should balance acquiring information from the unexplored feasible region and searching in promising areas of the design space. From a global optimization view, these phases are known as *exploration* and *exploitation* stages. The metaheuristic methods used different exploration phases allowing them to escape from local minima, but they are slower since they may not search the promising areas properly.

Hybridization is a common theme in research on metaheuristic algorithms to speed up the performance. As suggested by the "No Free Lunch Theorem" (Wolpert and Macready, 1997), no algorithm is best for solving all types of optimization problems. Consequently, we can expect problems with any one algorithm. When an algorithm fails to converge to the optimum or becomes too slow, a common resort is to hybridize it with one or more specially chosen algorithms. The idea is to combine the best features from different algorithms to come up with a more effective algorithm. This issue is presently undergoing intensive research and has been repeatedly shown to be more powerful and successful than the algorithms being hybridized. Some examples are Kirian and Gunduz (2013), Huang et al. (2013) and Wang and Si (2010), who showed hybridization is especially needed for solving high dimensional problems or in problems with multiple levels of optimization. In particular, Kirian and Gunduz (2013) showed that combining the global solutions from particle swarm optimization and artificial bee colony optimization (Karaboga, 2005) can provide superior or competitive performance when compared not only with either one of them alone but also with 12 other benchmark algorithms. Huang et al. (2013) hybridized continuous ant colony optimization (Dorigo, 1992; Socha and Dorigo, 2008) and particle swarm optimization to solve a data clustering problem and showed the hybridized version can prevent traps in local optima. A similar finding is reported in Wang and Si (2010) who hybridized a genetic algorithm and particle swarm optimization in mobile computing and showed the hybridized version required a shorter time to find the optimum than either one of the algorithms. Most recently, Wei and Tian (2017) hybridized particle swarm optimization and genetic algorithms for a more effective search for solutions to a multi-objective optimization problem.

Hybridized algorithms are used when it has been verified that they perform better than without the hybridization. The task is to select the best algorithm to hybridize with and usually this means finding an algorithm that has complementary properties with respect to the one another. Espinosa-Aranda et al. (2013) introduced a framework for the hybridization of metaheuristic algorithms. Table 4 shows a simplified version. Typically, an undetermined number of iterations is performed with an algorithm with good exploration properties, called $\mathcal{A}_G$, until N_c improvements of the objective function were reached. The other algorithm, say $\mathcal{A}_L$, with nice exploitation properties is then run N_r iterations to optimize the objective function. The strategy is to detect when the algorithm should change exploring the neighborhood. The hybridized algorithm continues to run the $\mathcal{A}_L$ method with good exploitative properties for a fixed number of iterations starting from the best current solution. This hybridization strategy has been applied to solve train time-tabling problems (Espinosa-Aranda et al., 2015) and segmentation of temporal series (López-García et al., 2014).

5. Numerical simulations

We now study the behavior and relative performances of various algorithms for finding different types of optimal designs for statistical models of increasing complexity. The objectives are to assess the viability and the relative merit of

Table 4
Hybridization of metaheuristic algorithms.

Step 1.	(*Initialization*). Let $\mathcal{A}_G$ and $\mathcal{A}_L$ be two optimization algorithms. Initialize the parameters of the algorithms $\mathcal{A}_G$ and $\mathcal{A}_L$ and the number of iterations (N). Fix the number of iterations N_c and N_r associated to $\mathcal{A}_G$ and $\mathcal{A}_L$, respectively. Set the counter t to 1 and s to 0 and let $\widehat{\Phi}^*_{aux} = +\infty$.
Step 2.	($\mathcal{A}_G$ *method*). Apply one iteration of the $\mathcal{A}_G$ algorithm to the current population. Let $\mathbf{z}^{t-1}$ be the current best solution and $\widehat{\Phi}^*$ its objective value. If $\widehat{\Phi}^*_{aux} > \widehat{\Phi}^*$, then let $s = s + 1$ and $\widehat{\Phi}^*_{aux} = \widehat{\Phi}^*$.
Step 3.	($\mathcal{A}_L$ *method*). If $s = N_c$ apply N_r iterations of the $\mathcal{A}_L$ algorithm starting from $\mathbf{z}^{t-1}$. Let $\mathbf{z}_L^{t-1}$ be the obtained solution, then replace the best solution by $\mathbf{z}^{t-1} = \mathbf{z}_L^{t-1}$ and set s to 0. If $s \neq N_c$, go to Step 4.
Step 4.	(*Stopping criterion*). If the current iteration is $t = N$, Stop; otherwise set $t = t + 1$ and go to Step 2.

the algorithms mentioned in the previous sections. To facilitate the use of these algorithms in practice, we implement computer codes on a user-friendly website to compare and assess quality of the different designs generated from various algorithms. Our intention here is not to identify a state-of-art algorithm that works best for finding all types of optimal designs but what types of algorithms are likely to perform better than others for finding optimal designs.

To this end, we have assessed several general purpose optimization programs. GAMS (The General Algebraic Modeling System) implements general purpose constrained optimization solvers for linear programming, mixed-integer programming, quadratic programming and nonlinear programming, which can be applied for large scale optimization problems. However, GAMS requires a formula to compute the inverse of the information matrix as a function of the input variables, which can be problematic for our purposes here. We chose MATLAB as a viable alternative because it is widely used and allows us to express the inverse of a matrix symbolically. Because our comparisons involve algorithms with different motivations, configurations and approaches, it is very hard to have an entirely fair comparison of their performance. This is more so for heuristic algorithms each of which can have very different numbers of tuning parameters and defining equations. For these reasons, in all our work, we used default values for the tuning parameters that came with the algorithms in MATLAB or in the original code.

We conducted three numerical simulations, each with a specific purpose. We work with selected statistical models and at the end of each numerical simulation, we present our results and observations. The three numerical simulations are:

- **Simulation 1:** The goal is to assess the viability of using metaheuristic algorithms to find optimal designs by analyzing their behavior, chance of convergence and whether hybridization improves their performance and accelerates convergence.
- **Simulation 2:** The goal is to assess the relative performance of both metaheuristic and exact algorithms for finding D- and A-optimal designs.
- **Simulation 3:** The goal is to evaluate which types of algorithms seem to be more useful for finding optimal designs for models with many factors.

Our main software is MATLAB R2017b and all numerical simulations are performed using a MSI Laptop with Processor Intel® Core™ i7-6700 HQ (2.6 GHz) and RAM 16 GB DDR4 2133 MHz.

5.1. Statistical models

In this section, we test and compare performances of several types of algorithms for finding various optimal designs using different models of varying complexity. To fix ideas, we focus on D- and A-optimality. Problem 1 concerns a design problem for a two-compartmental model widely used in pharmacokinetics and pharmacodynamics. Finding a locally D-optimal design for such a model is notoriously difficult because as the number of nonlinear parameters increases, the determinant of the information matrix tends to 0 (Dette et al., 2006). Problem 2 is a linear model with one factor included up to a quadratic term and another up to a linear term and an interaction term. The two factors are defined on different design spaces. Problems 3 and 12 are logistic models often used for studying a binary outcome and discussed in King and Wong (2000). The difference between these models is that one has 10 factors and the other has 3, the design space for each factor in the same model is the same but between models, the design spaces are different with one that is larger than the other. Problem 4 finds a c-optimal design to estimate the derivative of a function at a chosen point for the model discussed as Example 4.a in Yang et al. (2013) and from a more theoretical perspective in Dette and Melas (2011). Table 5 lists all statistical models with a few factors and Table 6 lists several generalized linear models with a larger number of factors. In this tables $g(\theta)$ is the function to be estimated.

Table 7 lists features of the 12 optimization problems and assumes that an upper bound of the number of support points in the optimal design is known before the number of variables to be optimized in the problem is calculated. It is not necessary to know the exact number of support points m^* but an upper bound m for this value should be known, (i.e. $m^* \leq m$). The algorithms calculate the weights for the m support points. The weights of the $m - m^*$ (unnecessary) support points are negligible (zero or near zero). Of course, in practice the number of support points m^* is unknown and

Table 5
Models with small or moderate number of factors.

Problem	Model	References
Problem 1	$Y \sim \theta_1 e^{-\theta_2 x} + \theta_3 e^{-\theta_4 x} + N(0, \sigma^2)$, $x \in [0, 3]$ $g(\theta) = (\theta_1, \theta_3)$ nominal values $\theta = (1, 1, 1, 2)^T$	Dette et al. (2006) and Example 1 in Yang et al. (2013);
Problem 2	$Y \sim \theta_1 + \theta_2 x_1 + \theta_3 x_1^2 + \theta_4 x_2 + \theta_5 x_1 x_2 + N(0, \sigma^2)$, $(x_1, x_2) \in [-1, 1] \times [0, 1]$ $g(\theta) = \theta$	Example 2 in Yang et al. (2013);
Problem 3	$Y \sim \pi_i(\mathbf{x}) = P(Y_i = 1\|\mathbf{x}) = \frac{e^{h(\mathbf{x})^T \theta_i}}{1+e^{h(\mathbf{x})^T \theta_1}+e^{h(\mathbf{x})^T \theta_2}} i = 1, 2$ $\mathbf{x} \in [0, 6]^3$ $h(\mathbf{x}) = [1, \mathbf{x}^T]^T$; $g(\theta) = (\theta_1^T, \theta_2^T)^T$ nominal values $\theta_1 = (1, 1, -1, 2)^T$; $\theta_2 = (-1, 2, 1, -1)^T$	Multinomial Logistic model with 3 categories and 3 factors as in Example 3 in Yang et al. (2013);
Problem 4	$Y \sim \theta_1 e^{\theta_2 x} + \theta_3 e^{\theta_4 x} + N(0, \sigma^2)$, $x \in [0, 1]$ $g(\theta) = \frac{\partial}{\partial x}\left(\theta_1 e^{\theta_2 x} + \theta_3 e^{\theta_4 x}\right)\|_{x=0} = \theta_1\theta_2 + \theta_3\theta_4$ nominal values $\theta = (1, 0.5, 1, 1)^T$	Example 4.*a* in Yang et al. (2013) and Dette and Melas (2011)
Problem 5	$Y \sim \frac{\theta_1\theta_3 x_1}{1+\theta_1 x_1+\theta_2 x_2} + N(0, \sigma^2)$ $(x_1, x_2) \in [0, 3] \times [0, 3]$ $g(\theta) = \theta$ nominal values $\theta = (2.9, 12.2, 0.69)^T$;	Kinetics of the catalytic dehydrogenation of n-hexil alcohol mode in Box and Hunter (1965)
Problem 6	$Y \sim \frac{\theta_1 x}{\theta_2+x} + N(0, \sigma^2)$ $x \in [0, 5]$ $g(\theta) = \theta$ nominal values $\theta = (1, 1)^T$;	Michaelis–Menten Model in López-Fidalgo and Wong (2002)
Problem 7	$Y \sim \frac{\theta_1 x_1}{\left(1+\frac{x_2}{\theta_3}\right)\theta_2+\left(1+\frac{x_2}{\theta_4}\right)x_1} + N(0, \sigma^2)$ $(x_1, x_2) \in [0, 30] \times [0, 60]$ $g(\theta) = \theta$ nominal values $\theta = (1, 4, 2, 4)^T$;	Mixed-type Inhibition Model in Bogacka et al. (2011)
Problem 8	$Y \sim \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_1 x_2 + \theta_5 x_1 x_3 + \theta_6 x_2 x_3 +$ $\frac{\theta_7}{x_1} + \frac{\theta_8}{x_2} + \frac{\theta_9}{x_3} + N(0, \sigma^2)$ $(x_1, x_2, x_3) \in [0.5, 2] \times [0.5, 2] \times [0.5, 2]$ $g(\theta) = \theta$	Three-factor model in Johnson and Nachtsheim (1983)

Table 6
Generalized linear models with a large number of factors.

Problem	Model	References
Problem 9	$Y \sim$ Binomial$(1, \mu)$; $\mu = \Phi(h(\mathbf{x})^T\theta)$ $\Phi(\cdot)$ is the normal cumulative distribution function $h(\mathbf{x}) = [1, \mathbf{x}^T]^T$; $\mathbf{x} \in [-2, 2]^5$ $g(\theta) = \theta$ nominal values $\theta = (0.5, 0.7, 0.18, -0.20, -0.58, 0.51)^T$;	Probit regression model with 5 factors
Problem 10	$Y \sim$ Binomial$(1, \mu)$; $\mu = \frac{1}{1+e^{-h(\mathbf{x})^T\theta}}$ $h(\mathbf{x}) = [1, \mathbf{x}^T]^T$; $\mathbf{x} \in [-2, 2]^5$ $g(\theta) = \theta$ nominal values $\theta = (0.5, 0.7, 0.18, -0.20, -0.58, 0.51)^T$;	Logistic regression model with 5 factors
Problem 11	$Y \sim \Gamma(1, \mu)$ $\mu = \left(\theta_1 x_1 + \sum_{i=2}^{5} x_{i-1} x_i \theta_i\right)^2$ $\mathbf{x} \in [0, 10]^5$ $g(\theta) = \theta$ nominal values $\theta = (0.25, 0.5, 0.20, 0.58, 0.51)^T$;	Gamma regression model with 5 factors and pairwise interaction terms
Problem 12	$Y \sim \pi_i(\mathbf{x}) = P(Y_i = 1\|\mathbf{x}) = \frac{e^{h(\mathbf{x})^T \theta_i}}{1+e^{h(\mathbf{x})^T \theta_1}+e^{h(\mathbf{x})^T \theta_2}} i = 1, 2$ $h(\mathbf{x}) = [1, \mathbf{x}^T]^T$; $\mathbf{x} \in [0, 3]^{10}$ $g(\theta) = (\theta_1^T, \theta_2^T)^T$ nominal parameters $\theta_1 = (1, 1, -1, 2, -2, 1, 0.5, -0.25, 0.5, -0.75, 2)^T$; $\theta_2 = (-1, 2, 1, -1, -1, -1, -0.5, 1, 0.75, 0.25, -2)^T$	Multinomial Logistic regression model with 10 factors

the algorithm finds it. As an illustration, Problem 9 in the table has 5 factors and, if we believe the number of support points required by the optimal design is $m = 25$, then there are 150 variables to optimize. This follows because there are 25×5 components in each of the support points to optimize along with the 25 weights.

5.2. Simulation 1: Compare performance of metaheuristics and their hybridized versions with the Nelder–Mead optimization strategy

In this simulation, we implement codes for SA, SA*, GA and PSO and compare their performances for finding optimal designs. The SA* algorithm has been shown to be fast and so we also select this modified SA algorithm to include in our comparison. We use the Standard Particle Swarm Optimization (SPSO) (Zambrano-Bigiarini et al., 2013) and find optimal designs using its code from Particle Swarm Central at www.particleswarm.info. The *Nelder–Mead Method (NM)* is a derivative-free method commonly used to optimize an objective function directly but it tends to converge to a local optimum. For our purpose here, we hybridize each of the metaheuristic algorithms with NM and we examine whether such hybridization improves performance. We denote their hybridized versions by SPSO+NM, GA+NM and SA+NM and SA*+NM. Note that the framework given in Table 4 is conceptual and it allows several alternatives for the algorithm $\mathcal{A}_L$. NM has been used in this numerical simulation but the SQP, IP or AM, among others can be used as $\mathcal{A}_L$. This type of hybridization is named *metaheuristic*.

Qiu et al. (2014) and Wong et al. (2015) found a variety of optimal designs using this algorithm. The GA algorithm we use is from the `GA` function in MATLAB. Similarly, our simulated annealing algorithm uses the `simulannealbnd` function in MATLAB.

We recall that SPSO, GA and SA (or SA*) are metaheuristics that do not guarantee convergence to the global optimum. In the context of design of experiments the typical *recombination* idea of GAs has the interpretation of exchanging support points among individuals in the population.

In what is to follow, we use the notation from Section 3.4 and for computational reasons let

$$\widehat{\mathbf{z}} = \max(\min(\mathbf{z}, \mathbf{U}), \mathbf{L}),$$

where the minimum and the maximum are computed component by component. This transformation projects the points out of the hyper-cube onto the boundary of the feasible region and allows us to use unrestricted optimization algorithms, such as the NM method, to tackle the constrained optimization problem (7). For the optimal weights, we first optimize a set of positive variables and then standardize them by dividing by their sum: $\widehat{\omega}_i = \omega_i / \sum_{i=1}^{m} \omega_i$. Such unconstrained optimization problems are generally easier to solve than constrained ones.

We solved each of the problems 10 times because of the stochastic nature of the algorithms and report the average results obtained. Throughout, we used the default values in `GA` and `simulannealbnd` subroutines in our computation. The NM method plays the role of the algorithm $\mathcal{A}_L$ in the hybridized algorithm (see Table 4) and the metaheuristic represents the algorithm $\mathcal{A}_G$. A key parameter used in the hybridization is the value of N_c. If the number of improvements obtained by the metaheuristic in previous iterates is equal to N_c then the NM is applied a number of iterations given from the best solution of the population (at the Step 3). The population is updated by interchanging the obtained solution by NM with the initial solution and this procedure is repeated. We used the values $N_c = 1$ and $N_c = 5$ and metaheuristic algorithms like SA/SA*, SPSO and GA based on populations of size small (5), medium (25), and large (125). Table 8 lists selected *D*- and *A*-optimal designs for our 12 problems. Each design has a horizontal line that separates the design support points from the weights. Each column represents a design point with one or more components, depending on the number of factors in the experiment. Fig. 2 shows the sensitivity function for the *D*-optimal design of problems 1 and 2 reported in Table 8, showing that the optimality conditions are satisfied.

Fig. 1 displays the evolution of the algorithms for generating *D*-optimal designs by plotting efficiencies of the generated designs from different algorithms versus the number of evaluations required. There are two hybridization parameters: N_c and the population size. We only present results of the best combination between the best choice of the value for the parameter N_c and the 3 population sizes for the Problems 1–4. The left hand side of the figure shows the performance of the basic metaheuristic algorithms and the right hand side shows the performance of their hybridized version for each of the 4 problems after 1000 or 5000 function evaluations. A similar plot for *A*-optimal designs exists but is omitted for space consideration. The efficiencies on the vertical axis in Fig. 1 were computed relative to the designs in Table 8.

Results from the left panel of the figure reveal that SA and its variant SA* do generally not perform as well as SPSO and GA. The latter two are generally competitive with GA performing better than SPSO for small-sized optimization problems. The right panel shows that results from the hybridized algorithms. They show SA still lags behind GA and SPSO when they are hybridized with NM and SA+NM still underperforms SA*+NM. There does not appear to be a clear winner between SPSO and GA when hybridized with NM, especially when results from the other problems are taken into account. There is however a clear trend that hybridization generally results in a more efficient design eventually with a possible exception in Problem 3. Our overall observations from Simulation 1 are the following:

1. The metaheuristic algorithms converge to the optimum and their rate of convergence is improved when they are hybridized with the NM method. This strategy is particularly useful when the algorithm converges slowly. Problem 3 is difficult (see Yang et al., 2013) and it demands more iterations. The hybridization strategy generally speeds up

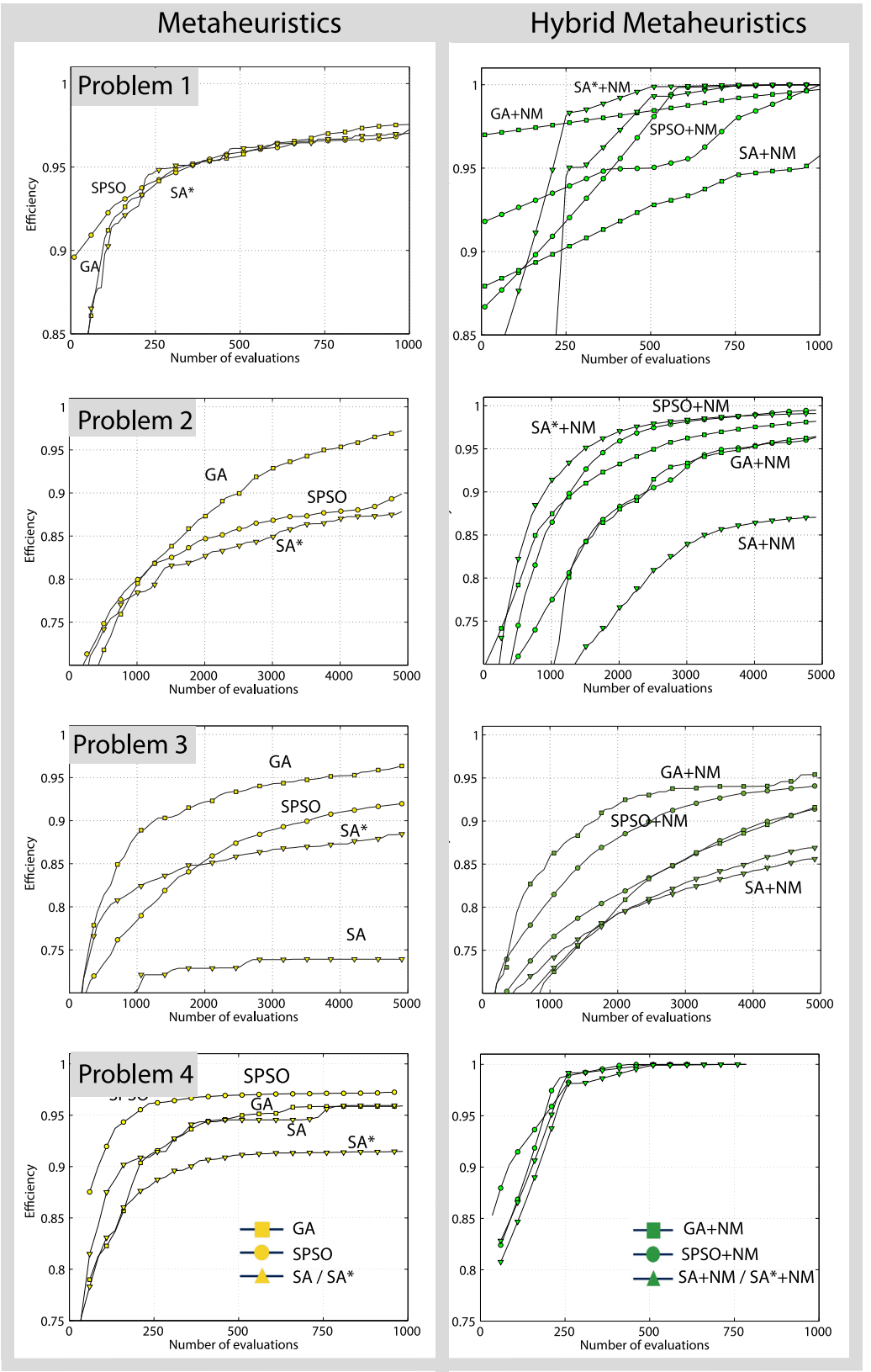


Fig. 1. Performance of the metaheuristic algorithms and their hybridized versions with the Nelder–Mead Method for finding *D*-optimal designs in Problems 1–4. The algorithms were run both for $N_c = 1$ and $N_c = 5$.

Table 7
Features of the 12 design problems when the optimal design is assumed to have a known number of support points.

Dimensionality	Problem	# factors	# parameters	# support points	# of variables to optimize
Small and moderate	1	1	4	6	12
	2	2	5	10	30
	3	3	8	15	60
	4	1	4	8	16
	5	2	3	10	30
	6	1	2	5	10
	7	2	4	5	15
	8	3	9	20	80
Large	9	5	6	25	150
	10	5	6	25	150
	11	5	5	25	150
	12	10	22	17	187

the metaheuristic methods SPSO and SA/SA* but not the GA. This may be because GA changes just one element of the population for the solution of the NM. This single individual is not essential in the evolution of the population due to the limited offspring derived from it. However, in SPSO, the changed solution is the best position that is used to update all particles.

2. No method clearly outperforms the others. This is not surprising due to the "No Free Lunch Theorem"; the GA is good for the linear model and the logit model (Problems 2 and 3) and the SPSO is a good alternative for the rest. SA and SA* are never the best alternatives.

5.3. Simulation 2: Comparing performances of algorithms using different models

In this numerical simulation, we apply and compare performances of general purpose algorithms with heuristic algorithms for computing *D*- and *A*-optimal designs with small to large numbers of factors. We use the term large to refer to problems with many factors and quite arbitrarily use 5 factors as the cutoff. As shown in Table 7, such regression models can become high dimensional quickly and we are then required to optimize many variables in the optimization problem.

Our algorithms of interest are:

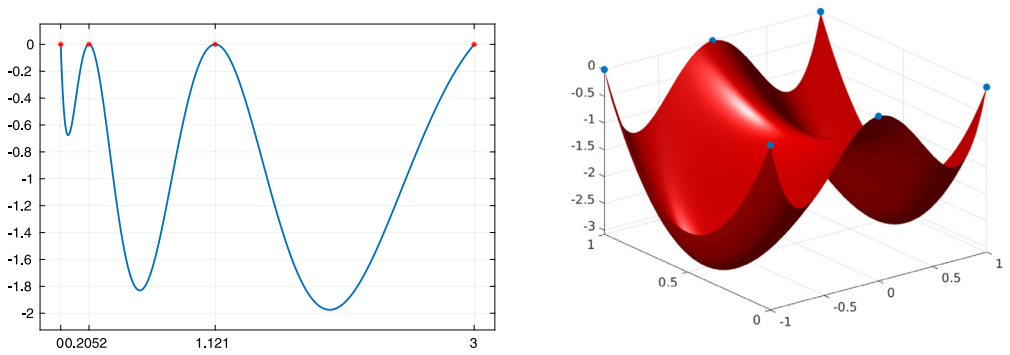
- **Exact algorithms.** We select the *Active-set method (ASM)*, the *Interior-point method (IPM)* and *Sequential Quadratic Programming (SQP)* in our comparison study. These algorithms are available in MATLAB and they are implemented in the `fmincon` function.
- **Heuristic algorithms.** We use SPSO and GA in our comparison study. For SPSO, we use the classic version and not PSO with an improved topology to help the algorithm converge to the global optimum. One reason is that our optimization problems are main convex optimization problems and so any local optimum found is also a global optimum.

We note that as the IPM searches within the interior of the feasible region, the solution should not lie on the boundary of the feasible region. To move iterates on the boundary to the interior, we introduced a perturbation to the current iterate by letting

$$\mathbf{z}' = \max\{\min\{\mathbf{z}, \mathbf{U} - \varepsilon\}, \mathbf{L} + \varepsilon\}, \tag{11}$$

where $\mathbf{z}$ is the current iterate. We chose $\varepsilon = 10^{-6}$ and $\mathbf{z}'$ is now an interior point in the feasible region near iterate $\mathbf{z}$. Our stopping rules for these algorithms are based on the number of function evaluations. For all problems we limit that number to 10 000. Each algorithm was run 25 times with different initial points and initial populations. In our work, all the tuning parameters in the algorithms GA, ASM, IPM and SQP are set to their default values in MATLAB. For this numerical simulation, several caveats were employed and worth keeping in mind; among them they are:

1. To standardize results across algorithms and facilitate comparison, we evaluate the efficiency lower bound of the design generated for each of our problems using Eq. (4). This means that we are required to find the global maximum of the directional derivative function of each generated design on the design space. This problem is non-convex with usually several local maxima and minima. In the literature, this problem is typically solved by computing the value of the function at each point of a user-selected grid on the design space. For problems when there are only 3 or fewer factors, such a procedure is likely to work satisfactorily. For example, we used a grid with 10 000 uniformly spaced points for our problems with one factor and for models with 2 or 3 factors, we increased the grid set to 50 000 uniformly spaced points in the design space. For problems with more than 3 factors, we no longer discretize the design space and use a few optimization methods to pick the maximum found. These methods include a second PSO (using the MATLAB function `particleswarm`), IPM, AS and SQP.



(a) Problem 1 (b) Problem 2

Fig. 2. Sensitivity function of Problems 1 and 2 at the *D*-optimal design.

2. For each method, repeated runs were conducted to find the maximum of the directional derivative function. The number of repeated runs depends on the size of the problems and the time it takes the search on average to determine the optimum. Our decision was to use 150 runs for each small/medium scale problem and 25 for each large scale problem. We then report the maximum value of the efficiency lower bounds found by the algorithm in the table.

Table 9 shows computation results for finding *D*- and *A*-optimal designs in Simulation 2. The first column shows the mean value of the objective function and the second column gives the efficiency lower bound attained by the design generated by each algorithm. The third column displays the average CPU time in seconds and the fourth column provides the average number of iterations performed. The extreme right column reports the percentage of successes in finding a design that has an efficiency bound greater than 50% and the algorithm converges.

We observe from Table 9 that exact methods tend to converge faster than heuristic algorithms. The exact algorithms converge to an optimal or highly efficient design usually in less than 1 s of CPU time for most of the problems we investigated. Among the exact algorithms, AS and SQP require less computing time than IPM in all cases. Problem 3 was posed by Yang et al. (2013) as an example of a design problem with many factors. They modified an existing algorithm and showed that it outperformed many classic methods on a discretized search space using a Dell Laptop with 2.2 GHz and 8 Gb RAM. It took 75 s when the grid size was 100^3 and about 600 seconds when the grid size was 200^3. They did not report efficiency lower bounds for the design found which may be because of the very large grid size. In contrast, our CPU times are decidedly smaller regardless of which of the 5 algorithms we used to find the optimal designs in Tables 10–11. We observe that both SPSO and GA require many more iterations which is not surprising because they are population based with many particles or genes. In contrast, the AS, SQP and IPM methods are all trajectory based that depend on a single initiated particle or point.

In the literature it has been highlighted that the SQP may suffer from ill conditioning problems and improvements, such as Morales et al. (2011), have been proposed to remedy it. Shahzad et al. (2012) emphasized that in each iteration of an IPM, a system of linear equations needs to be solved to find the search direction. This system becomes increasingly ill-conditioned as the IPM iterations converge. This effect has been numerically identified in the optimal experimental design problems. Results from our numerical computation suggest that SQP and AS are very sensitive to ill-conditioned problems and the IPM is least affected. In our work, some optimal designs have singular information matrices. This is common when we want to estimate a function of the model parameters. In these numerical tests, MATLAB reports warnings of *Matrix is singular, close to singular or badly scaled* producing failures in convergence. One way to handle the ill-conditioning problem is to add a small positive multiple of the identity matrix to the information matrix. For our case, we chose this positive constant to be $\delta = 10^{-10}$ and add $\delta \mathbf{I}_k$ to the information matrix. When the above trick is implemented, those errors stop occurring and convergence is obtained. The success percentage column shows that this strategy solves the ill conditioning issue most of the times. The upshot is that AM, SQP and IPM all seem to be good algorithms for finding efficient designs for models with a small to moderate number of factors. Recently, Esteban-Bravo et al. (2016) also showed that IPM performs well for finding exact optimal designs.

5.4. Simulation 3: High dimensional problems

We next compare the ability among general purpose algorithms to find optimal designs for this high dimensional optimization problem. Problems 9–11 involve 5 factors and Problem 12 involves 10 factors. Problem 12 is an extension of Problem 3 in Yang et al. (2013) from 3 to 10 factors. The state-of-the-art methods proposed by Yang et al. (2013) or Yu (2011) are not appropriate for this problem because they require the design space to be discretized. With many factors, the

Table 8
D- and *A*-optimal designs for Problems 1–7 and 11.

Problem	*D*-optimal designs
Problem 1	$\xi_{D_1} = \left[\begin{array}{cccc} 0.00000 & 0.20521 & 1.12108 & 3.00000 \\ \hline 0.50000 & 0.11774 & 0.14915 & 0.23311 \end{array}\right]$
Problem 2	$\xi_{D_2} = \left[\begin{array}{cccccc} -1.00000 & -1.00000 & -0.00000 & 0.00000 & 1.00000 & 1.00000 \\ 0.00000 & 1.00000 & 1.00000 & 0.00000 & 0.00000 & 1.00000 \\ \hline 0.18750 & 0.18750 & 0.12500 & 0.12500 & 0.18750 & 0.18750 \end{array}\right]$
Problem 3	$\xi_{D_3} = \left[\begin{array}{ccccccc} 0.00000 & 0.00000 & 0.00000 & 0.00000 & 1.55387 & 6.00000 & 6.00000 \\ 0.00000 & 2.42633 & 6.00000 & 6.00000 & 0.00000 & 0.00000 & 6.00000 \\ 0.00000 & 0.00000 & 3.25231 & 3.41470 & 0.00000 & 1.32188 & 5.41874 \\ \hline 0.19461 & 0.18966 & 0.05513 & 0.20825 & 0.18576 & 0.05324 & 0.11334 \end{array}\right]$
Problem 4	$\xi_{D_4} = \left[\begin{array}{cccc} 0.00000 & 0.33040 & 0.76914 & 1.00000 \\ \hline 0.24977 & 0.25019 & 0.25070 & 0.24934 \end{array}\right]$
Problem 5	$\xi_{D_5} = \left[\begin{array}{ccc} 0.28037 & 3.00000 & 3.00000 \\ 0.00000 & 0.00000 & 3.00000 \\ \hline 0.33333 & 0.33333 & 0.33333 \end{array}\right]$
Problem 6	$\xi_{D_6} = \left[\begin{array}{cc} 0.71429 & 5.00000 \\ \hline 0.50000 & 0.50000 \end{array}\right]$
Problem 7	$\xi_{D_7} = \left[\begin{array}{cccc} 3.15789 & 4.07934 & 30.00000 & 30.00000 \\ 0.00000 & 2.67542 & 0.00000 & 3.57895 \\ \hline 0.25000 & 0.25000 & 0.25000 & 0.25000 \end{array}\right]$
Problem 11	$\xi_{D_{11}} = \left[\begin{array}{ccccc} 0.00000 & 0.00000 & 0.00000 & 7.61183 & 9.93452 \\ 0.00000 & 3.14947 & 6.06868 & 10.00000 & 0.00000 \\ 10.00000 & 0.00000 & 9.61842 & 0.00000 & 0.15638 \\ 9.74706 & 8.98898 & 0.00000 & 0.00000 & 0.00000 \\ 0.00000 & 6.52986 & 8.86582 & 8.81591 & 0.02687 \\ \hline 0.20000 & 0.20000 & 0.20000 & 0.20000 & 0.20000 \end{array}\right]$
Problem	*A*-optimal designs
Problem 1	$\xi_{A_1} = \left[\begin{array}{cccccc} 0.00000 & 0.27119 & 0.78765 & 1.18375 & 1.21550 & 3.00000 \\ \hline 0.07671 & 0.18280 & 0.00000 & 0.28999 & 0.00000 & 0.45051 \end{array}\right]$
Problem 2	$\xi_{A_2} = \left[\begin{array}{cccccc} -1.00000 & -1.00000 & -0.00000 & -0.00000 & 1.00000 & 1.00000 \\ 0.00000 & 1.00000 & 1.00000 & 0.00000 & 0.00000 & 1.00000 \\ \hline 0.18591 & 0.13991 & 0.11966 & 0.22870 & 0.18591 & 0.13991 \end{array}\right]$
Problem 3	$\xi_{A_3} = \left[\begin{array}{ccccccccc} 0.00000 & 0.00000 & 0.00000 & 0.00000 & 0.00000 & 0.00000 & 1.47576 & 1.58577 & 6.00000 \\ 0.00000 & 0.00000 & 1.22431 & 2.89391 & 6.00000 & 6.00000 & 0.97114 & 0.00000 & 6.00000 \\ 0.00000 & 0.52498 & 0.00000 & 0.00000 & 3.18762 & 3.78094 & 0.00000 & 0.00000 & 5.52336 \\ \hline 0.23206 & 0.09296 & 0.08262 & 0.16852 & 0.02313 & 0.08555 & 0.06514 & 0.23019 & 0.01984 \end{array}\right]$
Problem 4	$\xi_{A_4} = \left[\begin{array}{cccc} 0.00001 & 0.30114 & 0.79254 & 1.00000 \\ \hline 0.18855 & 0.35107 & 0.31199 & 0.14839 \end{array}\right]$
Problem 5	$\xi_{A_5} = \left[\begin{array}{ccc} 0.20428 & 3.00000 & 3.00000 \\ 0.00000 & 0.00000 & 3.00000 \\ \hline 0.65062 & 0.27194 & 0.07744 \end{array}\right]$
Problem 6	$\xi_{A_6} = \left[\begin{array}{cc} 0.53727 & 5.00000 \\ \hline 0.66956 & 0.33044 \end{array}\right]$
Problem 7	$\xi_{A_7} = \left[\begin{array}{cccc} 2.44019 & 3.39189 & 30.00000 & 30.00000 \\ 0.00000 & 3.25160 & 0.00000 & 4.74092 \\ \hline 0.26507 & 0.32336 & 0.13982 & 0.27175 \end{array}\right]$
Problem 11	$\xi_{A_{11}} = \left[\begin{array}{ccccc} 0.00000 & 0.00000 & 0.00000 & 8.05372 & 9.58634 \\ 0.00000 & 0.65553 & 1.93390 & 10.00000 & 0.00000 \\ 4.59493 & 0.00000 & 6.03145 & 0.00000 & 8.92944 \\ 6.67427 & 6.40158 & 0.00000 & 0.00000 & 0.00000 \\ 0.00000 & 10.00000 & 9.23603 & 3.41055 & 9.23886 \\ \hline 0.28070 & 0.24682 & 0.09679 & 0.25408 & 0.12160 \end{array}\right]$

computer will require time to generate the grid set. Table 10 shows the amount of time required by a modern computer to generate the grid set when there are 10 factors and each of the 10 factor spaces is partitioned evenly into a number of

Table 9
Results for Simulation 2.

Problem	Algorithm	*D*-optimal design					*A*-optimal design				
		Φ_0	D-Eff	CPU (s)	# Iter.	Success	Φ_1	A-Eff	CPU (s)	# Iter.	Success
1	AS	253.31	100.00	0.06	233	96.0	19409.21	99.98	0.14	532	100.00
	SQP	253.31	100.00	0.06	303	100.0	19404.31	100.00	0.09	478	100.00
	IPM	253.31	100.00	0.24	818	100.0	19404.30	100.00	0.22	735	100.00
	SPSO	254.76	99.43	1.47	10 000	100.0	19440.83	99.81	1.59	10 000	100.00
	GA	254.20	99.65	1.67	10 050	100.0	19424.31	99.90	1.80	10 050	100.00
2	AS	2.73	100.01	0.09	376	100.0	4.1905	100.02	0.16	571	100.00
	SQP	2.74	99.81	0.11	534	100.0	4.1905	100.02	0.19	844	100.00
	IPM	2.75	99.34	0.42	1 932	100.0	4.2322	99.09	1.42	5 848	100.00
	SPSO	2.86	95.56	1.73	10 000	100.0	4.4531	94.20	2.00	10 000	100.00
	GA	2.79	97.86	2.22	10 050	100.0	4.3207	97.02	2.48	10 050	100.00
3	AS	9.14	96.43	5.04	1 703	100.0	35.5038	98.93	13.98	4 702	100.00
	SQP	9.23	95.66	6.53	2 224	100.0	35.6142	98.63	13.73	4 703	100.00
	IPM	8.82	99.66	27.77	9 580	100.0	35.4116	99.18	24.60	8 609	100.00
	SPSO	10.14	87.03	28.55	10 000	100.0	40.0745	87.85	28.46	10 000	100.00
	GA	10.08	87.26	28.94	10 050	100.0	42.9465	81.91	28.99	10 050	100.00
4	AS	193.62	99.01	0.06	248	92.0	2387200.47	98.49	0.18	678	76.00
	SQP	195.00	98.41	0.04	216	92.0	2434872.98	96.58	0.07	353	76.00
	IPM	192.51	99.54	0.21	641	92.0	2461270.95	96.28	0.26	756	96.00
	SPSO	192.04	99.78	1.51	10 000	100.0	2378230.42	98.85	1.62	10 000	100.00
	GA	191.77	99.92	1.71	10 050	100.0	2355457.84	99.79	1.83	10 050	100.00
5	AS	45.93	100.20	0.25	461	100.00	367.44	100.02	0.68	1 000	100.00
	SQP	45.93	100.20	0.28	532	100.00	367.44	100.02	0.59	1 006	100.00
	IPM	46.07	99.90	3.18	5 924	100.00	367.44	100.02	1.91	3 122	100.00
	SPSO	48.13	95.72	4.96	10 000	100.00	399.25	92.28	5.73	10 000	100.0
	GA	47.83	96.24	5.42	10 050	100.00	402.43	91.39	6.00	10 050	100.00
6	AS	13.82	100.00	0.02	96	100.00	40.09	100.00	0.04	159	100.00
	SQP	13.82	100.00	0.02	136	100.00	40.09	100.00	0.03	190	100.0
	IPM	13.84	99.92	0.14	660	100.00	40.11	99.94	0.26	1 134	100.00
	SPSO	13.82	100.00	1.20	10 000	100.00	40.09	100.00	1.33	10 000	100.00
	GA	13.82	100.00	1.41	10 050	100.00	40.09	100.00	1.60	10 050	100.00
7	AS	486.83	100.03	0.33	919	100.0	2467.83	100.01	0.37	1 002	100.00
	SQP	486.83	100.03	0.26	888	100.0	2486.60	99.30	0.32	1 089	100.00
	IPM	486.83	100.03	0.60	1 665	100.0	2468.02	100.00	0.71	2 017	100.00
	SPSO	694.43	73.43	2.66	10 000	84.0	3683.66	69.44	2.67	10 000	52.00
	GA	490.24	99.34	3.00	10 050	100.0	2553.74	96.70	2.90	10 050	100.00
8	AS	3.14	91.04	0.74	2 072	100.00	11.98	82.02	1.32	3 052	100.00
	SQP	3.14	91.10	0.97	2 965	100.00	11.98	82.02	1.39	4 106	100.00
	IPM	3.18	89.79	3.13	9 108	100.00	12.06	81.55	3.39	9 778	100.00
	SPSO	4.09	70.40	3.17	10 000	100.00	17.80	55.43	3.21	10 000	76.00
	GA	3.77	75.93	4.22	10 050	100.0	16.09	61.24	4.33	10 050	100.0

Table 10
CPU times in seconds required to generate a set of uniformly spaced points for 10 factors.

Points per factor space	Number of points of the grid, N	CPU to generate the grid (s)
2	$2^{10} = 1024$	0.0067
3	$3^{10} = 59\,049$	0.2302
4	$4^{10} = 1\,048\,576$	3.1136
5	$5^{10} = 9\,765\,625$	27.5529
6	$6^{10} = 60\,466\,176$	172.2832
7	$7^{10} = 282\,475\,249$	848.2922

points. We note that a simple refinement of 7 points for each of the 10 factors' space will require a grid with more than 282 million points and about 15 minutes to generate the grid. If the design criterion is not differentiable, additional time is required to compute the subgradient of the criterion at each candidate design and the computational time becomes prohibitive. This suggests that promising algorithms like the YBT–Newton method proposed by Yang et al. (2013) may become problematic for high dimensional design problems because it requires the design space to be discretized.

There is clearly high computational cost for these problems and so we have just ran 5 runs per algorithm and limited number of iterations allowed to be 50 000. For problem 12, the efficiency bounds are quite low and we remove the constraint that the efficiency bound has to yield at least 50% before it is removed to ensure a timely success.

Table 11
Results for comparing general purpose and metaheuristic algorithms for solving high-dimensional design problems with several factors.

Problem	Algorithm	*D*-optimal design					*A*-optimal design				
		Φ_0	D-Eff	CPU (s)	# Iter.	Success	Φ_1	A-Eff	CPU (s)	# Iter.	Success
9	AS	0.82	86.87	35.31	8 154	100.0	1.25	92.17	69.53	16 097	100.0
	SQP	0.82	86.81	34.36	7 762	100.0	1.24	92.32	91.24	20 506	100.0
	IPM	0.93	77.28	193.33	44 239	100.0	1.26	91.37	215.90	49 785	100.0
	SPSO	0.95	74.83	217.23	50 000	100.0	1.40	82.02	216.76	50 000	100.0
	GA	0.81	87.87	225.24	50 050	100.0	1.27	90.17	225.37	50 050	100.0
10	AS	1.90	94.77	9.82	7 279	100.0	2.67	96.14	22.53	16 853	100.0
	SQP	1.93	93.44	10.58	8 034	100.0	2.66	96.26	22.06	16 610	100.0
	IPM	2.15	84.34	62.03	45 475	100.0	2.94	88.37	67.19	49 373	100.0
	SPSO	2.13	84.52	75.84	50 000	100.0	2.98	86.22	65.48	50 000	100.0
	GA	1.99	94.82	83.23	50 050	100.0	2.68	95.49	75.07	50 050	100.0
11	AS	0.18	100.00	5.57	4 863	100.0	0.2139	99.80	10.23	9 026	80.0
	SQP	0.18	100.00	5.18	4 621	100.0	0.2135	100.00	11.83	10521	100.0
	IPM	0.18	99.88	23.26	20 373	100.0	0.2135	99.98	49.86	43 632	100.0
	SPSO	0.29	61.38	55.00	50 000	100.0	0.3075	70.12	55.74	50 000	100.0
	GA	0.19	93.07	63.22	50 050	100.0	0.2278	93.75	62.89	50 050	100.0
12	AS	4.99	61.14	51.96	15 175	100.0	14.1631	34.27	149.84	43560	100.0
	SQP	4.92	61.97	61.87	18 473	100.0	13.9885	34.69	163.99	49 230	100.0
	IPM	4.77	63.88	153.23	45 762	100.0	15.8348	30.71	168.83	50 039	100.0
	SPSO	7.85	38.91	165.89	50 000	100.0	26.9684	18.08	164.59	50 000	100.0
	GA	5.80	52.46	176.57	50 050	100.0	19.5341	24.87	175.41	50 050	100.0

Table 11 reports CPU times and success rates for the 5 algorithms for finding *A*-optimal and *D*-optimal designs for models with a large number of factors. In Simulation 2 there appear to be no clear differences in the performances among the exact algorithms. For high dimensional optimization problems, Simulation 3 has 3 takeaways: (i) AS and SQP are the most efficient with respect to CPU time, (ii) metaheuristic methods have a very low convergence rate and they require much more time to a solution, and (iii) all the designs have a large efficiency but for Problem 12. We believe that for Problem 12 the design is efficient although the quality of the bound is rather low. This is because for obtaining this bound the directional derivative is optimized over the design space. If this space is high dimensional, the maximum found for the sensitive function can be unstable or sensitive.

5.5. Computational and numerical issues

There are 3 main issues we encountered when running these algorithms. The first is that sometimes the generated information matrix is either singular or very close to being singular. This has been discussed for the Simulation 2 above and may be applied in general. Moreover, the success of the algorithm depends very much on the initial points and the random points generated during the process. This problem was recently addressed by Huang et al. (2019).

The second problem from the practical point of view is that the algorithms tend to produce clusters of support points at some locations. Additionally, it is not clear each point found with a very small weight is a support point of the optimal design or it should be merged with others and considered as one point. This requires a post-processing rule that specifies how these clusters of points should be merged into a single point or not. The number of support points chosen is greater than those of the optimal design. For this reason, the solution obtained at some steps has pairs of points very near each other with significant weights. In the iterative procedure, such near points are replaced by their mean and the corresponding weight is obtained summing the two weights. A threshold has to be tuned in the algorithm for determining when two points are considered near enough.

A third problem is in the numerical computation of the efficiency lower bound, which requires solving the optimization problem (4). Fig. 2 shows the sensitivity function for Problems 1 and 2, they are non-concave functions, causing that the maximizing of this function is a hard task. If the optimization algorithm converges to a local optimum or some other point, the efficiency lower bound will be incorrect. For this reason, we tackled the optimization problem (4) with a few algorithms PSO, SQP, IPM and ASM and the best of the four solutions was chosen. If the problem has three or less factors the optimization is performed over a grid set.

We have devoted a considerable amount of effort to develop a user-friendly MATLAB tool, called ODEm, that implements the algorithms described in this paper for computing *A*- and *D*-optimal approximate designs, and more. This software has also been used for computing the numerical simulations. This MATLAB tool is available in a MethodsX companion paper (García-Ródenas et al., 2019). Moreover, the ODEm installer, the source files .m and the manual can be downloaded from github.com/JoseAngelMartinB/ODEm.

6. Conclusions

Our work compares a few commonly used general purpose optimization algorithms for finding *A*- and *D*-optimal experimental designs. The algorithms include exact methods and an increasingly popular class of algorithms called nature

inspired metaheuristic algorithms. Previous literature reviews on algorithms were less ambitious, either focusing on exact methods or confined to finding only one type of optimal designs, or optimal designs for linear models only, as in Cook and Nachtsheim (1980), Haines (1987), Meyer and Nachtsheim (1995) and Nguyen and Miller (1992).

Comparing the performance of algorithms can be problematic in many ways that have not been well discussed in the literature. The important question to consider in such comparative work is whether algorithms are fairly compared. Even among deterministic algorithms, such skepticism was recently echoed by Meng (2014), who questioned whether there can be a truly fair comparison between algorithms. Comparing metaheuristic algorithms is especially difficult because they generally come with very different motivations and one iteration may have a very different meaning in another algorithm. Further, there is the constant question that if a metaheuristic algorithm under-performs another, it may just be that the right set of tuning parameters was not used. One option is to tackle this issue to employ the recommended default values that came with the original version of metaheuristic algorithms, which is what we did in this paper. Even then, there is ongoing intensive work in coming up with more effective choice of tuning parameters for different metaheuristic algorithms. Of course, a compounding issue that may further defy a fair comparison across these algorithms is that they come with different number of tuning parameters. So, while our work represents a first step in comparing performances across different types of algorithms to find optimal designs, there are some clear limitations in our work.

The general purpose optimization algorithms are desirable because they search for the optimal designs without having the need to compute the sub-gradients, when applicable, or analyze the convergence properties of the generated designs in the previous iterations. Using several models and optimality criteria, we have shown that heuristic algorithms such as GA, SPSO and SA and their hybridizations generally converge to the *D*-optimal or *A*-optimal designs. For the types of problems we have considered, the GA and the SPSO+NM algorithms appear to have better performance in terms of computing efficiency for finding the optimal designs. Such results are especially helpful when the theoretical construction of the optimal designs is not available. While the general-purpose optimization algorithms can be applied successfully to solve many types of optimization problems, they may come with conditions such as requiring that the objective function is differentiable or that the dimension of the optimization problem is not too large. Otherwise, they are powerful and easily available for implementation after modifying them for the purpose at hand. For example, in our work, we modified the IPM algorithm from MATLAB and we showed it has a good performance for finding *A*- and *D*-optimal approximate designs and also for obtaining an approximate optimal design for a nonlinear model with 10 factors. Esteban-Bravo et al. (2016) used the IPM to find exact optimal designs for linear regression models and reached a similar conclusion.

There are two key conclusions from our paper. First, for exact algorithms, the SQP and AS are very sensitive to ill-conditioned problems and frequently result in practical non-convergence due to numerical problems. This ill-conditioned problem can sometimes be avoided by adding a small positive definite matrix to the Fisher information matrix before inversion, in which case, both SQP and AS become very competitive and can find nearly optimal designs in less than one second when the problems has a small or moderate number of factors. The IPM is generally less sensitive to ill-conditioning issues and has a more robust convergence. Second, for metaheuristic algorithms, our results suggest that the exact methods SQP, AS and IPM outperform metaheuristic algorithms like GA and SPSO in both efficiency and CPU time. We also note that the rate of convergence of metaheuristic algorithms tends to be slower than that from exact methods. The numerical results, in Simulation 1, have demonstrated that the hybridization strategy accelerates the convergence of metaheuristic algorithms. For small to moderate sized problems, both types of algorithms find an optimal design in a reasonable CPU time.

In summary, our view is that when the optimization problem is complex and no theory exists for finding and verifying the optimality of a design, it is helpful to use various nature-inspired metaheuristic algorithms with different sets of tuning parameters and hope that they all converge to the same design. When this happens, the solution is most likely the optimal design that we sought.

Acknowledgments

The research of Wong reported in this paper was partially supported by the National Institute of General Medical Sciences of the National Institutes of Health, USA under the Grant Award Number R01GM107639. López-Fidalgo and Wong were sponsored by Spanish Research Agency and fondos FEDER, Spain MTM2016-80539-C2-R1. Wong wishes to acknowledge the support from the University of Castilla–La Mancha, Spain jointly with the program FEDER of Castilla–La Mancha 2007–2013 that made his visit possible and he thanks the department for the warm hospitality during the visit. The research of García-Ródenas and Martín-Baos was supported by Ministerio de Economía, Industria y Competitividad — FEDER EU, Spain grant with number TRA2016-76914-C3-2-P. The research of García-García was supported by the predoctoral FPU fellowship from the Ministerio de Educación, Cultura y Deportes, Spain with number 16/00792.

The contents in this paper are solely the responsibility of the authors and do not necessarily represent the official views of the National Institutes of Health. We also give thanks to Yu Shi from UCLA and Ping-Yang Chen from National Cheng Kung University in Tainan, Taiwan for checking codes in the algorithms provided to generate optimal designs. No grants from private companies have been received for this study. We would like to express our thanks to the two reviewers for their extremely valuable comments which have allowed us to improve the paper.

References

Amo-Salas, M., Delgado-Marquez, E., Lopez-Fidalgo, J., 2016. Optimal experimental designs in the flow rate of particles. Technometrics 58 (2), 269–276.

Atwood, C.L., 1976. Convergent design sequences, for sufficiently regular optimality criteria. Ann. Statist. 4 (6), 1124–1138.

Berger, M., Wong, W., 2009. An Introduction to Optimal Designs for Social and Biomedical Research. John Wiley and Sons.

Bogacka, B., Patan, M., Johnson, P.J., Youdim, K., Atkinson, A.C., 2011. Optimum design of experiments for enzyme inhibition kinetic models. J. Biopharm. Statist. 21 (3), 555–572.

Box, G.E.P., Hunter, W.G., 1965. The experimental study of physical mechanisms. Technometrics 7 (1), 23–42.

Broudiscou, A., Leardi, R., Phan-Tan-Luu, R., 1996. Genetic algorithm as a tool for selection of D-optimal design. Chemometr. Intell. Lab. Syst. 35, 105–116.

Černý, V., 1985. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. J. Optim. Theory Appl. 45 (1), 41–51.

Chaloner, K., Larntz, K., 1989. Optimal Bayesian design applied to logistic regression experiments. J. Statist. Plan. Inf. 21, 191–208.

Chen, R.B., Hsieh, D.N., Hung, Y., Wang, W., 2013. Optimizing latin hypercube designs by particle swarm. Stat. Comput. 23 (5), 663–676.

Chen, R.B., Yang, P., Wong, W.K., 2017. Standardized maximin D-optimal designs for pharmacological models via particle swarm optimization techniques. Chemometr. Intell. Lab. Syst. (169), 79–86.

Chernoff, H., 1953. Locally optimal designs for estimating parameters. Ann. Math. Stat. 24 (4), 586–602.

Cook, R.D., Nachtsheim, C.J., 1980. A comparison of algorithms for constructing exact D-optimal experimental designs. Technometrics 22, 315–324.

Cuervo, D.P., Goos, P., Sorensen, K., Arraiz, E., 2014. An iterated local search algorithm for the vehicle routing problemwith backhauls. European J. Oper. Res. 237 (2), 454–464.

Dette, H., Melas, V.B., 2011. A note on the De la Garza phenomenon for locally optimal designs. Ann. Statist. 39 (2), 1266–1281.

Dette, H., Melas, V.B., Wong, W.K., 2006. Locally D-optimal designs for exponential regression models. Statist. Sinica 16, 789–803.

Dorigo, M., 1992. Optimization, Learning and Natural Algorithms (Ph.D. thesis). Politecnico di Milano.

Espinosa-Aranda, J.L., García-Ródenas, R., Angulo, E., 2013. A framework for derivative free algorithm hybridization. In: Adaptive and Natural Computing Algorithms, Vol. 7824. Springer, pp. 80–89.

Espinosa-Aranda, J.L., García-Ródenas, R., Ramírez-Flores, M., López-García, M., Angulo, E., 2015. High-speed railway scheduling based on user preferences. European J. Oper. Res. 246 (3), 772–786.

Esteban-Bravo, M., Leszkiewicz, A., Vidal-Sanz, J.M., 2016. Exact optimal experimental designs with constraints. Stat. Comput. 21 (4), 1–19.

Fedorov, V., 1972. Theory of Optimal Experiments. Academic Press.

García-Ródenas, R., García-García, J.C., López-Fidalgo, J., Martín-Baos, J.Á., Wong, W.K., 2019. ODEm: A Matlab tool for finding optimal approximate experimental designs. MethodsX (submitted for publication).

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning, first ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Haines, L.M., 1987. The application of the annealing algorithm to the construction of exact optimal designs for linear-regression models. Technometrics 29, 439–447.

Hamada, M., Martz, H.F., Reese, C.S., Wilson, A.G., 2001. Finding near-optimal Bayesian experimental designs via genetic algorithms. Amer. Statist. 55 (3), 175–181.

Hassan, R., Cohanim, B., de Weck, O., Venter, G., 2005. A comparison of particle swarm optimization and the genetic algorithm. In: 46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference. American Institute of Aeronautics and Astronautics.

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems, second ed. MIT Press, Cambridge, MA, USA, 1992.

Huang, Y., Gilmour, S.G., Mylona, K., Goos, P., 2019. Optimal design of experiments for non-linear response surface models. J. R. Stat. Soc. Ser. C. Appl. Stat. 68 (3), 623–640.

Huang, C.L., Huang, W.C., Chang, H.Y., Yeh, Y.C., Tsai, C.Y., 2013. Hybridization strategies for continuous ant colony otimization and particle swarm optimization applied to data clustering. Appl. Soft Comput. 13, 3864–3872.

Ingber, L., 1993. Simulated annealing: Practice versus theory. Math. Comput. Modelling 18 (11), 20–57.

Johnson, M.E., Nachtsheim, C.J., 1983. Some guidelines for constructing exact D-optimal designs on convex design spaces. Technometrics (25), 271–277.

Karaboga, D., 2005. An Idea Based on Honey Bee Swarm for Numerical Optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department.

Kennedy, J., Eberhart, R., 1995. Particle swarm optimization. In: IEEE International Conference on Neural Networks - Conference Proceedings, Vol. 4, pp. 1942–1948.

Kiefer, J., 1974. General equivalence theory for optimum designs (Aproximate theory). Ann. Statist. 2 (5), 848–879.

Kim, S., Erwin, D., Wu, D., 2012. Efficacy of dual lung cancer screening by chest x-ray and sputum cytology using Johns Hopkins Lung Project Data. Biometrics Biostat. 3, 1000139.

Kim, S., Li, L., 2011. A novel global search algorithm for nonlinear mixed-effects models using particle swarm optimization. J Pharmacokinet. Pharmacodyn. 38, 471–495.

King, J., Wong, W.K., 2000. Minimax D-optimal designs for the logistic model. Biometrics 56, 1263–1267.

Kirian, M.S., Gunduz, M., 2013. A recombination-based hybridization of particle swarm optimization and artificial bee colony algorithm for continuous optimization problems. Appl. Soft Comput. 13, 2188–2203.

Kirpatrick, S., Gelatt, C.D., Vechi, M.P., 1983. Optimization by simulated annealing. Science 220 (4598), 671–680.

Latif, A.H.M.M., Brunner, E., 2016. A genetic algorithm for designing microarray experiments. Comput. Statist. (2), 409–424.

Leatherman, E., Dean, A., Santner, T., 2014. Computer experiment designs via particle swarm optimization. Springer Proc. Math. Stat. 114, 309–317.

Lin, C.D., Anderson-Cook, C.M., Hamada, M.S., Moore, L.M., Sitter, R.R., 2015. Using genetic algorithms to design experiments: A review. Qual. Reliab. Eng. Int. 31 (2), 155–167.

López-Fidalgo, J., Wong, W.K., 2002. Design issues for the Michaelis–Menten model. J. Theoret. Biol. 215 (1), 1–11.

López-García, M.L., García-Ródenas, R., Gómez, A.G., 2014. Hybrid meta-heuristic optimization algorithms for time-domain-constrained data clustering. Appl. Soft Comput. 23, 319–332.

Lu, Z., Pong, T.K., 2013. Computing optimal experimental designs via interior point method. SIAM J. Matrix Anal. Appl. 34, 1556–1580.

Lukemire, J., Mandal, A., Wong, W.K., 2019. d-QPSO: A quantum-behaved particle swarm technique for finding D-optimal designs with discrete and continuous factors and a binary response. Technometrics 61 (1), 77–87.

Mak, S., Joseph, V., 2018. Minimax and minimax projection designs using clustering. J. Comput. Graph. Statist. 27 (1), 166–178.

Mak, S., Wu, C.F.J., 2019. Analysis-of-marginal-tail-means (ATM): A robust method for discrete black-box optimization. Technometrics 0, 1–15.

Mandal, A., Wong, W.K., Yu, Y., 2015. Algorithmic searches for optimal designs. In: Handbook of Design and Analysis of Experiments. Chapman and Hall, pp. 89–98.

Marini, F., Walczak, B., 2015. Particle swarm optimization (PSO): A tutorial. Chemometr. Intell. Lab. Syst. 315–323.

Martín-Martín, R., Torsney, B., López-Fidalgo, J., 2007. Construction of marginally and conditionally restricted designs using multiplicative algorithms. Comput. Statist. Data Anal. 51, 5547–5561.
Meng, X.L., 2014. Response: Did Newton–Raphson really fail? Stat. Methods Med. Res. 3 (23), 312–314.
Meyer, R., Nachtsheim, C.J., 1995. The coordinate-exchange algorithmm for constructing exact optimal experimental designs. Technometrics 37, 60–69.
Morales, J.L., Nocedal, J., Wu, Y., 2011. A sequential quadratic programming algorithm with an additional equality constrained phase. IMA J. Numer. Anal. 32 (2), 553–579.
Nelder, J.A., Mead, R., 1965. A simplex method for function minimization. Comput. J. 7, 308–313.
Nguyen, K., Miller, A.J., 1992. A review of some exchange algorithms for constructing discrete D-optimal designs. Comput. Statist. Data Anal. 14, 489–498.
Pazman, A., 1986. Foundations of Optimum Experimental Design, Vol. 14. Springer.
Qin, A.K., Huang, V.L., Suganthan, P.N., 2009. Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans. Evol. Comput. 13 (2), 398–417.
Qiu, J., Chen, R.B., Wang, W., Wong, W.K., 2014. Using animal instincts to design efficient biomedical studies via particle swarm optimization. Swarm Evol. Comput. 18, 1–10.
Royle, J.A., 2002. Exchange algorithms for constructing large spatial designs. J. Statist. Plann. Inference 100, 121–134.
Sengupta, S., Basak, S., Peters, R.A., 2018. Particle swarm optimization: A survey of historical and recent developments with hybridization perspectives. Mach. Learn. Knowl. Extr. 1 (1), 157–191.
Shahzad, A., Kerrigan, E.C., Constantinides, G.A., 2012. A stable and efficient method for solving a convex quadratic program with application to optimal control. SIAM J. Optim. 22 (4), 1369–1393.
Silvey, S.D., 1980. Optimal Design. Chapman & Hall.
Smith, K., 1918. On the standard deviations of adjusted and interpolates values of an observed polynomial functions and its constants and the guidance they give towards a proper choice of the distribution of observations. Biometrika 12, 1–85.
Socha, K., Dorigo, M., 2008. Ant colony optimization for continuous domains. European J. Oper. Res. 185 (3), 1155–1173.
Stallings, J.W., Morgan, J.P., 2015. General weighted optimality of designed experiments. Biometrika 102 (4), 925–935.
Su, P.L., Chen, Y.S., 2012. Implementation of a genetic algorithm on MD-optimal designs for multivariate response surface models. Expert Syst. Appl. 39, 3207–3212.
Sun, W., Yuan, Y.X., 2006. Optimization Theory and Methods: Nonlinear Programming. Springer.
Wang, L., Si, G., 2010. Optimal location management in mobile computing with hybrid genetic algorithm and particle swarm optimization (GA-PSO). In: 2010 17th IEEE International Conference on Electronics, Circuits and Systems, pp. 1160–1163.
Wei, W., Tian, Z., 2017. An improved multi-objective optimization method based on adaptive mutation particle swarm optimization and fuzzy statistics algorithm. J. Stat. Comput. Simul. 87, 2480–2493.
Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Trans. Evol. Comput. 1, 67–82.
Wong, W.K., Chen, R.B., Huang, C.C., Wang, W., 2015. A modified particle swarm optimization technique for finding optimal designs for mixture models. PLoS One 10 (6), e0124720.
Wynn, H.P., 1972. Results in the theory and construction of D-optimum experimental designs. J. R. Stat. Soc. Ser. B Stat. Methodol. 73, 133–147.
Yang, X.S., 2010. Nature-Inspired Metaheuristic Algorithms. Luniver Press.
Yang, M., Biedermann, S., Tang, E., 2013. On optimal designs for nonlinear models: A general and efficient algorithm. J. Amer. Statist. Assoc. 108 (504), 1411–1420.
Yang, J., Tong, L., Mandal, A., 2017. D-optimal designs with ordered categorical data. Statist. Sinica 27 (4), 1879–1902.
Yu, Y., 2011. D-optimal designs via a cocktail algorithm. Stat. Comput. 21 (4), 475–481.
Zambrano-Bigiarini, M., Clerc, M., Rojas, R., 2013. Standard particle swarm optimisation 2011 at CEC-2013: A baseline for future PSO improvements. In: Evolutionary Computation (CEC), 2013 IEEE Congress on, pp. 2337–2344.