21

Algorithmic Searches for Optimal Designs

Abhyuday Mandal, Weng Kee Wong, and Yaming Yu

CONTENTS

21.1 Introduction

Research in optimal experimental design has a long history and dates back as early as 1918 in a seminal paper by Smith (1918) and probably earlier. This chapter discusses algorithms for finding an optimal design given a statistical model defined on a given design space. We discuss background and the need for algorithms to find an optimal design for various situations. There are different types of algorithms available in the literature, and even for the same design problem, the researcher usually has several algorithms to choose from to find an optimal design. There are also algorithms that use specialized methods to find an optimal design for a very specific application. For example, Syed et al. (2011) used a mathematical programming technique to search for a D-optimal design using cyclotomic cosets. The literature on algorithms to find an optimal design for a statistical model is therefore huge and diverse.

The aim of this chapter is to give a brief overview on algorithms for finding optimal designs and to discuss selected algorithms that represent some of the current trends. We also highlight algorithms that are more widely applicable for solving different types of design problems. We discuss typical problems encountered in using an algorithmic approach to find an optimal design and present alternative algorithms from other fields that seem capable of generating efficient designs quickly and easily for any model and objective in the study. Along the way, we provide pseudocodes for some of these algorithms and illustrate how they work to solve real and current design problems in biomedicine.

In recent years, experimental costs have risen steeply at many levels, and researchers increasingly want to minimize study costs without having to sacrifice the quality of the statistical inference. Even with cost considerations aside, design issues are still very important. This is because a poor design can provide unreliable answers either because the estimates have unacceptably large variances or it provides low power for testing the main hypothesis in the scientific study. In the extreme case, when the study is so badly designed, it may not even provide an answer to the main scientific question of interest no matter how large the sample size. Thus all studies should be carefully planned at the onset. The main goal or goals have to be clearly specified in advance, along with all model assumptions, constraints, and practical concerns associated with execution and interpretation of the experiment.

Typically, a mathematical function called a design optimality criterion is formulated to reflect the objectives of the study as accurately as possible. A common criterion is D-optimality for estimating all model parameters; if only a subset of the model parameters is of interest, D_s-optimality is used. Both criteria seek to minimize the volume of the confidence region for the parameters of interest when errors are independent and normally distributed. Other commonly used design criteria are discussed in other chapters of this book. In general, given the user-selected criterion and a statistical model, the design problem is to find the optimal distinct combinations of the independent variables that define the treatments for which the outcome is to be observed and the number of replicates (or repeats) at each of these settings. Throughout the chapter, we assume the sample size n is predetermined, and so we are not dealing with a sample size determination problem.

This chapter is organized as follows. In the next section, we discuss the need for a carefully designed study in a real biomedical problem and briefly describe how design issues can be addressed using an algorithm described later in the chapter. Section 21.3 provides background material, fundamental concepts in designs, common terminology, and tools to find an optimal design as well as verify if a design is optimal. Section 21.4 reviews two popular types of algorithms: Fedorov–Wynn type of algorithms and exchange algorithms. In Section 21.5, we discuss alternative and modern algorithms for generating an optimal design, and in Section 21.6, we present metaheuristic algorithms, which have more recently been used as effective and practical tools for finding optimal designs. A summary is offered in Section 21.7.

21.2 Algorithmic Approach to Solve a Design Problem: A Motivating Example

This section provides an example of a real problem that can be solved using an algorithm discussed in this chapter. We omit technical details but provide references for the source where the solution of the problem, codes, and implementation of the actual algorithm can be found. The motivating application we have in mind is how to efficiently design a drug discovery study.

21.2.1 Drug Discovery Problem

Identifying promising compounds from a vast collection of feasible compounds is a challenging problem in drug discovery. In this combinatorial chemistry problem, the goal is to obtain sets of reagents (or monomers) that maximize certain efficacy of a compound. However, here, the objective is to identify several "nearly best" combinations, rather than only one "best" or optimal one. In a typical problem, a core molecule is identified to which reagents are attached at multiple locations. Each attachment location may have tens or hundreds of potential monomers. Mandal et al. (2009) considered an example where a compound was created by attaching reagents to the three locations of a molecule, denoted by A, B, and C (e.g., see Figure 21.1). In this kind of application, the compound library (the set of all feasible compounds) may consist of 5 feasible substructures (monomers) at position A, 35 at position B and 250 at position C. That is, the compound library has a total of $5 \times 35 \times 250 = 43{,}750$ chemical compounds. Production of all these compounds for physical testing is expensive, and thus, it is desirable to select a relatively much smaller subset of the compounds with desirable properties. Once a compound is created, its physiochemical properties (namely, absorption/administration, distribution, metabolism, excretion, toxicity [ADMET]) are used to identify whether the compound is promising or not.

Using terminology in the design literature, we have here three factors (A, B, and C) in this design problem with 5, 35, and 250 levels, respectively. The response can be one of the ADMET properties of a compound, identified by a particular combination of A, B, and C. Alternatively, we can use a multiple-objective design discussed in Chapter 25 to capture the goals of the study simultaneously. The purpose of this study is to identify level combinations that can reduce, say, the toxicity of a compound, and at the same time increase, say, its absorption capability. Mandal et al. (2007) used desirability scores to reduce the dual goals to a single-objective optimization problem, where the goal was to identify a compound (i.e., a design point) $\boldsymbol{x}_i$ that will maximize the objective function ψ given by

$$\psi(\boldsymbol{x}_i) = \psi(x_{i1}, \ldots, x_{ip}). \tag{21.1}$$

Here, $p = 3$ and (x_{i1}, x_{i2}, x_{i3}) denote the levels of the three factors A, B, and C. Common design techniques such as fractional factorial designs, orthogonal arrays, and response surface designs have been widely used in screening studies for a long time in many industries (Dean and Lewis 2006). In this problem, the outcome or outcomes do not have a known mean structure in terms of the factors, and so they cannot be applied directly. With the

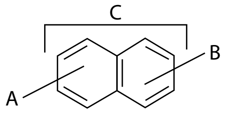

FIGURE 21.1
The core molecule of a compound with three reagent locations. (Reprinted (adapted) with permission from Mandal, A., Johnson, K., Wu, C.F.J., and Bornmeier, D., Identifying promising compounds in drug discovery: Genetic algorithms and some new statistical techniques, *J. Chem. Inf. Model.*, 47, 981–988. Copyright 2007 American Chemical Society.)

advent of technology, more complicated assumptions are increasingly required, and consequently, nonstandard design approaches are called for to solve the design problem. There is no analytical description of the optimal design, and so an optimal design has to be found using an algorithm. In this case, we applied a metaheuristic algorithm to search for an optimal design.

Metaheuristic algorithms are gaining in popularity because of their many successes in tackling high-dimensional optimization problems in other fields. The research in this field is very active; there are many book chapters and monographs in this area, including some that are updated in less than 2 years just to keep track of its rapid development; see Yang (2008, 2010) for example. Metaheuristic algorithms are based on heuristics that are, in part or all, not supported by theory. Here, we take the term *heuristic* to mean any procedure, or algorithm, that is not wholly supported by theory but appears to work well in practice. For example, in the algorithm used by Cook and Nachtsheim (1982) described just before Section 21.4.1, the authors offer a heuristic rule calling for restarting their algorithm after every 25 iterations for greater efficiency. Such rules are provided based on empirical experience and may not even apply to other problems or scenarios. The prefix meta- in metaheuristic suggests that it has a common mission and not a specific one for a particular problem. In our case, a metaheuristic algorithm for finding an optimal design means no specific feature of the design problem should greatly affect the functionality of the algorithm. For instance, the algorithm should work whether the design criterion is differentiable or not or whether the criterion is convex or not. Within broad limits, performance of the algorithm should not depend on the number of design variables, and constraints on those variables should be easily accommodated. Generally, only obvious modifications of the algorithm are required, and they will include, for example, modifying the statistical model or the design criterion or the design space. This is in contrast to, say, Fedorov-type algorithms where it is assumed that the optimality criterion is differentiable. Of course, as expected, more complicated optimization problems will require more time to solve.

Mandal et al. (2006) used a version of the genetic algorithm (GA), which is a metaheuristic algorithm, to find the optimal compounds to be created in the laboratory. In their application, $p = 3$ and x_{i1}, x_{i2}, and x_{i3} take the possible values of A, B, and C, respectively. The ψ in (21.1) is user selected and an example of such a function is given in (21.9). In that section, we illustrate how GA can be applied to identify the settings that maximize the objective function.

21.3 Background

Throughout this chapter, we assume that we are given a model, an optimality criterion, and a fixed sample size n and the problem is how to take n independent observations from the given design space in an optimal way. When a parametric approach is adopted, the *ith* outcome y_i is modeled by

$$y_i = f(\boldsymbol{x}_i, \boldsymbol{\beta}) + \epsilon_i, \quad i = 1, \ldots, n. \tag{21.2}$$

Here, ϵ_i is the error incurred at the ith trial, and all errors are independent, and each is distributed as $\epsilon_i \sim N(0, \sigma^2)$. The mean response function $f(\boldsymbol{x}, \boldsymbol{\beta})$ is assumed to be known, and $\boldsymbol{\beta}$ is the vector of unknown parameters. All outcomes are observed at user-selected points from a specified compact design space $\mathcal{X}$, which can be multidimensional. The choice of the user-selected points $x_1, x_2, \ldots, x_n \in \mathcal{X}$ makes up the design for the study.

Linear models are commonly used in scientific studies. They arise as a special case of (21.2) when the mean response function can be written as the product of two components with $f'(x_i, \beta) = f'(x_i)\beta$ and $f(x)$ is the regression function with p linearly independent components. In this chapter, linear models with heteroscedastic, independent, and normal errors are called general linear models and defined by

$$y_i = f'(x_i)\beta + \epsilon_i, \quad i = 1, \ldots, n. \tag{21.3}$$

We follow optimal design terminology and use an efficiency function $\lambda(x)$ to incorporate the heteroscedasticity by letting $\mathrm{Var}(\epsilon_i) = \sigma^2/\lambda(x_i)$. When the efficiency function is known, design issues can be directly addressed using a suitably transformed homoscedastic model.

Following convention, the goodness of a design is measured by the Fisher information matrix. Apart from an unimportant multiplicative constant, this matrix is obtained by first computing the expectation of the matrix of second derivatives of the log likelihood function at a single point and then averaging it over all the design points and multiplying by -1. For nonlinear models, this matrix depends on the unknown parameters β, but not so for linear models. For example, if (21.3) holds and we have resources to take n_i observations at $x_i, i = 1, 2, \ldots, k$, then the information matrix for this linear model is

$$\sum n_i f(x_i) f'(x_i),$$

apart from a multiplicative constant. When the model is nonlinear, we approximate the information matrix by replacing $f(x_i)$ by the gradient of $f(x_i, \beta)$ in the aforementioned matrix. The simplest way to construct optimal designs for nonlinear models is to assume that nominal values for β are available. After plugging the nominal values into the information matrix, the criterion no longer depends on β, and the design problem can be solved using design techniques for linear models. Because such optimal designs depend on the nominal parameter values, they are termed locally optimal. In what is to follow, the information matrix is denoted by $\boldsymbol{M}(\xi, \beta)$ where ξ is the design used to collect the data. If we have a linear model, the information matrix does not depend on β, and we simply denote it by $\boldsymbol{M}(\xi)$.

There are two types of optimal designs: approximate optimal designs and exact optimal designs. Approximate designs, or continuous designs, are easier to find and study than exact optimal designs. Approximate designs are essentially probability measures defined on a compact and known design space. In our setting, we assume that we are given a preselected sample size n, a design criterion, and a statistical model. The design questions are how to choose design points in the design space to observe the outcome in an optimal manner. For approximate designs, the optimization problem finds the optimal probability measure that then has to be rounded appropriately to an exact design before it can be implemented in practice. For example, if n is the total number of observations to be taken in the experiment and the approximate design calls for taking observations at three points x_1, x_2, and x_3 from $\mathcal{X}$ with weights w_1, w_2, and w_3, the implemented design takes nw_i observations at $x_i, i = 1, 2, 3$ such that $nw_1 + nw_2 + nw_3 = n$, and the weights are rounded so that each of the summands is an integer. Clearly, the implemented design is not unique as it depends on the rounding procedure. In contrast, an optimal exact design solves the optimization problem by finding an optimal integer variable k, the number of unique design points, where these points $x_1, x_2, \ldots, x_k$ are from the design space and the number of observations $n_1, n_2, \ldots, n_k$ to be taken at each of these design points and subject to the constraint that they sum to n.

The main appeal of working with approximate designs is that after formulating the objective as a convex function of the information matrix, we have a convex optimization problem, and equivalence theorems from convex analysis theory can be directly used to verify whether a design is optimal or not (Kiefer 1974). In addition, algorithms are available for finding different types of optimal approximate designs. There is no general algorithm for finding an optimal exact design and no general analytical tool for confirming whether an exact design is optimal or not.

The traditional way of finding an optimal design is a mathematical derivation based on model assumptions. Frequently, tedious algebra and specialized mathematical tools that exploit special properties of the regression functions are required to determine the optimal design. For simpler problems, a closed form formula for the optimal design may be possible, but at other times only an analytical description is available. Generally, D-optimal designs are considered the easiest to find compared with other types of optimal designs. For example, if we wish to estimate only a subset of the whole set of model parameters in the mean function, formulae can be complicated. Furthermore, a D_s-optimal design for estimating selected parameters in a polynomial regression model is usually described by a system of equations whose solutions provide the canonical moments of the optimal design, and the optimal design is then recovered from the set of canonical moments in a complex manner; details are given in a monograph by Dette and Studden (1997).

Having a formula or a closed-form description of the optimal design is helpful because it facilitates studying properties of the optimal design, including its sensitivities to model assumptions. However, a purely theoretical approach to describing an optimal design analytically can be limiting in terms of scope and usability. This is because the optimal design is derived under a very strict set of assumptions, and so the theoretical results are applicable to that specific setting only. For instance, model assumptions frequently made at the onset of the study may be questionable, and it is desirable to know how robust the optimal design is when some aspects of the model assumptions are changed. Chapter 20 elaborates on this important issue and how to make a design more robust to model assumptions. As a specific case, errors are often assumed to have a known homoscedastic structure for simplicity. What is the corresponding optimal design when errors become slightly heteroscedastic? Unfortunately, the optimal design for the model with heteroscedastic errors frequently cannot be deduced from the theory used for the construction of the optimal design for the homoscedastic model because the technical justifications used in the derivation of an optimal design are usually quite involved and not applicable for another model. This is especially true for nonlinear models where the method of proof usually depends uniquely on the model and the design criterion. Consequently, analytical results stated for a particular situation are of limited use in practice where different conditions apply.

Algorithms are therefore very useful for generating different types of optimal designs in practice because they can overcome the problems just described associated with the theoretical approach. Algorithms generally do not depend on the mathematical complexities of the problem as much as analytical approaches, and at the same time, they can also apply more generally to different settings. Users provide design inputs for the problems of interest, and the algorithm converges to the optimal design or a design close to the optimum. If the criterion is differentiable, Fedorov-type algorithms have been shown to converge to the optimal design for a linear model (Fedorov 1972). If the criterion is not differentiable, we know of no algorithm that has been proven to converge to the optimal design for a general linear or nonlinear model. An example of a class of nondifferentiable criteria is the minimax type of criteria, where the optimal design minimizes the maximum of some quantity and the minimization is over all designs associated with the defined space. For example, in response surface estimation, one may want to find a design that minimizes the maximum variance of the fitted response over a user-selected region. Another example of a minimax criterion concerns parameter estimation, where there are variances from a few estimated parameters in the model, and the goal is to find a design that minimizes the maximum of these variances. A sample of work in the construction of minimax optimal designs for linear models can be found in Wong (1992), Wong and Cook (1993), Brown and Wong (2000), and Chen et al. (2008). King and Wong (2000) constructed minimax optimal designs for the two-parameter logistic model when the nominal value of each of the two parameters was known to belong to a specified interval, and the goal was to find a minimax D-optimal design that minimized the maximal inefficiency that could arise from misspecification of the nominal values from the two intervals. Using similar ideas, Berger et al. (2000) found minimax optimal designs for the more complicated item theory response models commonly used in education research. Most recently, Chen et al. (2014) proposed a nature-inspired metaheuristic algorithm for finding minimax optimal designs in a general nonlinear regression setup.

21.4 Review of Selected Traditional Algorithms

This section provides a review of a few algorithms to find an optimal design. Because there are many algorithms in the field, we only discuss selected methods and provide a list of references from the literature.

With continued advances in technology, computational cost has decreased rapidly, and exhaustive search is becoming feasible for some problems thought to be *prohibitively large* even a decade ago. However, with the advancement of science, the demand for more complex designs has gone up as well. One example is in the construction of optimal designs for event-related fMRI studies in Chapter 25. Finding alternatives to an exhaustive search is always desirable, and some of the old algorithms, after suitable modifications, have reemerged and have been shown to be quite successful in recent years. For example, Ranjan et al. (2008) used the branch-and-bound algorithm proposed by Welch (1982) and found sequential optimal designs for contour estimation using complex computer codes. Yang et al. (2015) used Fedorov-type exchange algorithm, originally published in 1969 (see also Fedorov 1972), to obtain optimal designs for generalized linear models. There are several versions of the exchange algorithms where the search begins with a single random design, and then each design point is considered for exchange with other points. The pair of points chosen for exchange is the pair that results in maximum gain of the optimality criterion. Similarly, the "DETMAX" algorithm proposed by Mitchell (1974) may be considered an early version of an exchange algorithm. Atkinson et al. (2007) discussed variants of the exchange algorithms.

To delineate properties of the algorithms, it is helpful to compare their performances under a broad variety of settings and identify situations where some may outperform others. This is typically a hard job as one has to carefully define the scope and choose appropriate algorithms to compare using several well-defined measures of goodness. Cook and Nachtsheim (1980) compared several algorithms for constructing exact discrete D-optimal designs, and Johnson and Nachtsheim (1983) gave guidelines for constructing exact D-optimal designs. Nguyen and Miller (1992) gave a comprehensive review of some exchange algorithms for constructing exact discrete D-optimal designs.

Applications of computer algorithms to find an optimal design are abundant in the literature, and several of them can be found in our reference list. One such example is given in Cook and Nachtsheim (1982), where they wanted to estimate uranium content in calibration standards. They assumed the underlying model is a polynomial of degree 1–6 and modified the Fedorov's algorithm described in the succeeding text to find an optimal design capable of providing maximal information on the uranium density along the uranium log. The optimal design provides useful guidelines for how the alternating sequence of thin and thick disks should be cut from the uranium–aluminum alloy log before the thin disks are used for destructive analyses.

21.4.1 Fedorov–Wynn Type of Algorithms

The Fedorov–Wynn type of algorithm is one of the earliest and most notable algorithms for finding optimal approximate designs that has enjoyed and continues to enjoy widespread popularity. In the most basic form, the algorithm requires a starting design and a stopping rule. It then proceeds by adding a point to the current design to form a new design and repeats this sequence until it meets the condition of the stopping rule. There are several modifications of the original algorithm currently in use; they may be tailored to a particular application or modified to speed up the convergence. A main reason for its popularity is that this is one of the few algorithms that can be proved to converge to the optimal design if it runs long enough. Many subsequent algorithms in the literature for finding optimal designs have features in common with the original Fedorov–Wynn algorithm.

As an illustration, we describe here the essential steps in the Fedorov–Wynn type of algorithm for finding a D-optimal approximate design for a general linear model in (21.3) where $\text{Var}(y(\boldsymbol{x}))$ is $\sigma^2/\lambda(\boldsymbol{x})$ so that $\lambda(\boldsymbol{x})$ is inversely proportional to the known variance of the response at the point $\boldsymbol{x} \in \mathcal{X}$. Technical details of the algorithms including proof of its convergence can be found in Fedorov (1972).

Pseudocode for Fedorov–Wynn algorithm:

```
1. Set $t = 0$ and choose a starting approximate design $\xi_t$ with a nonsingular information matrix.
2. Compute its information matrix $\boldsymbol{M}(\xi_t)$ and obtain its inverse $\boldsymbol{M}^{-1}(\xi_t)$.
3. Determine the point $\boldsymbol{x}^*$ that maximizes $\lambda(\boldsymbol{x}^*)d(\boldsymbol{x}^*, \xi_t)$ over all $\boldsymbol{x}$ in the design space,
   where $d(\boldsymbol{x}, \xi_t) = f'(\boldsymbol{x}^*)\boldsymbol{M}^{-1}(\xi_t)f(\boldsymbol{x}^*)$ is the variance of the fitted response at the
   point $\boldsymbol{x}^*$, apart from an unimportant multiplicative constant.
4. Generate a new design $\xi_{t+1} = (1-\alpha_t)\xi_t + \alpha_t \nu_{\boldsymbol{x}^*}$ where $\alpha_i'$s is a preselected sequence
   of numbers between 0 and 1 such that its limit is 0 and its sum is infinite. Here, $\nu_{\boldsymbol{x}^*}$
   is the one point design supported at the point $\boldsymbol{x}^*$.
5. If the stopping rule is not met, replace $t$ by $t + 1$ and go to step 2.
```

A more specific choice for α_t is possible to increase the efficiency of the algorithm. For example, one may choose α_t to maximize the increase in the determinant of the current information matrix at each iteration. It can be shown using simple calculus that this leads to the choice of $\alpha_t = \zeta_t/[(\zeta_t + p - 1)p]$ where $\zeta_t = \lambda(\boldsymbol{x}^*)d(\boldsymbol{x}^*, \xi_{t-1}) - p$ and p is the number of the parameters in the mean function. In the last step, the stopping rule is user specified.

Two common examples of stopping rules are the maximum number of iterations allowed and when $\lambda(\boldsymbol{x}^*)d(\boldsymbol{x}^*, \xi_t) - p < \kappa$ for some small prespecified positive κ. The term $\lambda(\boldsymbol{x})d(\boldsymbol{x}, \xi_t)$ appears frequently when we study D-optimal designs for heteroscedastic models and is sometimes called the weighted variance of the response at the point $\boldsymbol{x}$ using design ξ_t.

21.4.2 Exchange Algorithms

In this section, we review several exchange strategies for exact designs. Some of these are analogues of Fedorov–Wynn algorithms of the previous section for exact designs. Consider the general optimization problem where the objective function is $\psi(x_1, \ldots, x_n)$ and one wishes to maximize $\psi(\cdot)$ with respect to $\boldsymbol{x}_j \in \mathcal{X}$, $j = 1, \ldots, n$, where $\mathcal{X}$ is the design space. Again, let us use D-optimal designs for linear regression as an example. Suppose an observation is taken at the design point $\boldsymbol{x}_i$, and the outcome is modeled as $y_i = f'(\boldsymbol{x}_i)\boldsymbol{\beta} + \epsilon_i$, where $\epsilon_i \sim \mathrm{N}(0, \sigma^2)$ and all errors are independent. D-optimality corresponds to maximizing $\psi(x_1, \ldots, x_n) = \det[f'(\boldsymbol{X})f(\boldsymbol{X})]$ where $\boldsymbol{X}' = (x_1, \ldots, x_n)$ and $f'(\boldsymbol{X}) = (f(x_1), \ldots, f(x_n))$. In other words, each design point $\{\boldsymbol{x}_j\}$ corresponds to a row of the model matrix $f(\boldsymbol{X})$.

Exchange algorithms iteratively modify the current design by deleting existing design points and adding new points from the design space $\mathcal{X}$ in an effort to increase the design criterion ψ. Multiple runs with different starting designs are often performed due to issues with local maximizers. Well-known algorithms that fall in this category include Wynn's (1972) algorithm, Fedorov's (1972) algorithm and its modification (Cook and Nachtsheim 1980), and k-exchange algorithms (Johnson and Nachtsheim 1983). A basic step in these algorithms is exchanging a point $\boldsymbol{x}_i$ in the current design with some $\boldsymbol{x}_* \in \mathcal{X}$ where $\boldsymbol{x}_*$ is chosen such that the improvement in the objective function ψ is the greatest. They differ, however, in the choice of $\boldsymbol{x}_i$. For Fedorov's algorithm, $\boldsymbol{x}_i$ is chosen such that the improvement in ψ after the exchange is the greatest among all $\boldsymbol{x}_i$. Thus, each exchange effectively performs n optimizations, one for each $\boldsymbol{x}_i$ in the current design, but only implements the best of these exchanges in the next design. Cook and Nachtsheim (1980) propose a modified Fedorov algorithm, where each iteration performs n exchanges, one for each $\boldsymbol{x}_i$. The following are basic pseudocodes for these algorithms:

Pseudocode for Fedorov algorithm:

```
1. Choose the initial design $(x_1^0, x_2^0, \ldots, x_n^0)$.
2. At iteration $t$, suppose the current design is $(x_1^t, x_2^t, \ldots, x_n^t)$.
   a. For $1 \le j \le n$, compute $\psi_j \equiv \max_{\boldsymbol{x}} \psi(x_1^t, \ldots, x_{j-1}^t, x, x_{j+1}^t, \ldots, x_n^t)$ where the
      maximization is over all $x \in \mathcal{X}$. Let $x_j^*$ be the corresponding maximizer.
   b. Find $j^* = \arg\max_j \psi_j$. Set $x_k^{t+1} = x_k^*$, $k = j^*$ and $x_k^{t+1} = x_k^t$, $k \ne j^*$.
3. Stop when there is no appreciable improvement in $\psi$.
```

Pseudocode for modified Fedorov algorithm:

```
1. Choose the initial design $(x_1^0, x_2^0, \ldots, x_n^0)$.
2. At iteration $t$, suppose the current design is $(x_1^t, x_2^t, \ldots, x_n^t)$. For $j = 1, \ldots, n$ in turn,
   set $x_j^{t+1} = \arg\max_{\boldsymbol{x}} \psi(x_1^{t+1}, \ldots, x_{j-1}^{t+1}, x, x_{j+1}^t, \ldots, x_n^t)$ where the maximization is over
   all $x \in \mathcal{X}$.
3. Stop when there is no appreciable improvement in $\psi$.
```

In the k-exchange method, a subset of k current design points is chosen for exchange (replacement). These k points are often chosen so that deletion of each results in the smallest k decreases in ψ. More sophisticated algorithms include the DETMAX algorithm of Mitchell (1974). For some design criteria, these algorithms exploit special relationships between ψ before and after the exchange. For example, for D-optimality, when a design point $\boldsymbol{x}_i$ is replaced by another point $\boldsymbol{x}_* \in \mathcal{X}$, the multiplicative change in ψ is given by

$$\Delta(\boldsymbol{x}_i, \boldsymbol{x}_*) = 1 + v(\boldsymbol{x}_*, \boldsymbol{x}_*) - v(\boldsymbol{x}_i, \boldsymbol{x}_i) + v^2(\boldsymbol{x}_i, \boldsymbol{x}_*) - v(\boldsymbol{x}_*, \boldsymbol{x}_*)v(\boldsymbol{x}_i, \boldsymbol{x}_i), \tag{21.4}$$

where $v(\boldsymbol{a}, \boldsymbol{b}) = f'(\boldsymbol{a})\boldsymbol{M}^{-1}f(\boldsymbol{b})$ and $\boldsymbol{M}$ is the information matrix before the exchange, $\boldsymbol{M} = f'(\boldsymbol{X})f(\boldsymbol{X})$. There is also a simple formula to compute $\boldsymbol{M}^{-1}$ after each exchange without any full matrix inversion.

Meyer and Nachtsheim (1995) proposed a cyclic coordinate-exchange algorithm and showed its effectiveness on several common design criteria such as the D-criterion and the linear criteria (i.e., linear functionals of the inverse information matrix). Rodriguez et al. (2010) used a similar strategy for G-optimal designs. In its basic form, the cyclic coordinate-exchange algorithm works as follows. Suppose each design point can be written as a vector with p coordinates, and suppose the design space $\mathcal{X}$ is the Cartesian product of the corresponding p subspaces. Writing the design criterion as

$$\psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \equiv \psi(x_{11}, \ldots, x_{1p}, \ldots, x_{n1}, \ldots, x_{np}), \tag{21.5}$$

where $\boldsymbol{x}_i \equiv (x_{i1}, \ldots, x_{ip}) \in \mathcal{X}$, we iteratively optimize ψ over each x_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, p$ in turn. Thus, the method is an example of the widely used cyclic ascent algorithm. Here, each x_{ij} may be discrete or continuous. For a continuous one-dimensional x_{ij}, one can use various optimization routines such as golden-section search or parabolic interpolation (Rodriguez et al. 2010). One cycle of the algorithm consists of a sequence of np optimizations, one over each variable x_{ij}. The algorithm is stopped either when there is not much improvement in ψ after the latest cycle or when a prespecified number of cycles have been performed. As with other exchange methods, multiple runs with different starting values are recommended.

21.4.3 Issues with Algorithms

Many algorithms proposed in the literature typically proceed sequentially as follows. The user first inputs quantities that define the design problem. They typically include the statistical model, the design space, and the optimality criterion. If the model is nonlinear, the user would also have to supply nominal values for the model parameters before running the algorithm. The user then provides a (nonsingular) starting design, and the algorithm iterates until the stopping criterion is satisfied. For approximate designs, the procedure typically iterates by mixing the current design with a specially selected point to form a new design; this can be done simply by taking a convex combination of the design and the point using a sequence of weights that converges but not prematurely. An example of such a sequence of weights is to use $1/n$ at the nth iteration. For a differentiable design criterion, such as D-optimality, the selected point to introduce at each iteration is the point that maximizes the weighted variance function. The user also specifies a stopping criterion to tell the algorithm when to stop; this can be in terms of either the maximum number of iterations allowed or the minimum change in the criterion value over successive iterations. These steps are clearly exemplified in the Fedorov–Wynn type of algorithms and its many modifications. Technical details including proof of their convergence can be found in design monographs such as Fedorov (1972), Silvey (1980), and Pázman (1986). Cook and Nachtsheim (1980) provide a review and comparisons of performance of various algorithms in different settings.

With algorithms, the main issues are proof of convergence, speed of convergence, ease of use and implementation, how applicable they are to find optimal designs for different types of problems, and how their performance compares to that of competing algorithms. Therefore, care should be exercised in the selection of an appropriate algorithm. For example, the Fedorov–Wynn type of algorithms introduces a point to the current design at each iteration, and frequently the generated design has many support points clustered around the true support points of the optimal design. Sometimes, periodic collapsing of these clusters of points to single points can accelerate convergence. Typical rules for collapsing the clusters of points into single points may be applied after a certain number of iterations, say, 100, with the expectation that this number may vary from problem to problem. If the rule for collapsing accumulated points is not appropriate, it may take a longer time to find the optimal design. Some algorithms, such as particle swarm optimization (PSO)-based algorithms described later on in the chapter, do not have this issue. However, mathematical proof of convergence of Fedorov–Wynn type of algorithms is available, but none exists for particle swarm–based algorithms.

Another issue with algorithms is that they may get *stuck* at some design and have difficulty in moving away from it in the direction of the optimal design. This can happen randomly or in part because of a poor choice of the starting design. Further, even though an algorithm has been proven to converge for some types of models, the result may not hold for others. For example, we recall the proof of convergence of Fedorov–Wynn type of algorithms was given for linear models only. When we apply them to nonlinear models or models with random effects, the algorithm may not work well.

A distinguishing feature of approximate designs is that when the criterion is a convex function of the information matrix, it is possible to verify whether the generated design is optimal or not. When the design space is low dimensional, a graphical plot can confirm optimality using convex analysis theory. The same theory also provides a lower bound for the efficiency of the generated design if it is not optimal. The case with exact optimal designs is very different. Frequently, there is no guarantee that the design generated by the algorithm is indeed the optimal one because a general tool for confirming optimality is not available. Researchers often show that their proposed algorithm works better by producing a design that is superior to those obtained from other algorithms.

21.5 Alternative Algorithms for Finding Optimal Designs

As mentioned before, there are other numerical tools for finding optimal designs. For example, Chaloner and Larntz (1989) used Nelder–Mead method to search for Bayesian A-, c-, and D-optimal designs for logistic regression models with one independent variable. Purely exchange-type algorithms are known to perform extremely poorly in this context. One can also rewrite the approximate design problem as an unconstrained optimization problem and apply the popular conjugate gradient or quasi-Newton methods to solve the resulting optimization problem (Atkinson et al. 2007). These are powerful and general optimization methods, which can be applied to find various types of optimal designs for a wide variety of models and design criteria. However, they may not be the best ways for finding approximate designs (Yu 2011) because they can be quite slow and convergence is not guaranteed. We elaborate on this a bit in Section 21.5.1.

Another class of methods that has recently gained popularity is mathematical programming methods. Some of these methods were used as early as the 1970s but only more recently have statisticians started investigating how such methods work for finding optimal designs. Examples of such methods include semidefinite programming (SDP) that is well discussed in Vandenberghe and Boyd (1996) and semi-infinite programming (SIP) that is well discussed in Reemtsen and Ruckman (1998). In SDP, one optimizes a linear function subject to linear constraints on positive semidefinite matrices. Depending on the design criteria, this may not be an easy task. For example, it is relatively easy for A- or E-optimality, where the optimization problem can be rewritten as an SDP and then solved using standard methods such as interior point methods. Atashgah and Seifi (2009) discussed SDP for handling optimal design for multiresponse experiments. The examples provided by the authors were quite illuminating, and the approach seemed more versatile and powerful than the algorithm proposed by Chang (1997) for finding D-optimal designs for multiple response surface models constructed from polynomials. See also Papp (2012) and Duarte and Wong (2014a) who applied SDP to find optimal designs for rational polynomial models and several types of Bayesian optimal design problems, respectively. Filová et al. (2012) also applied SDP to find another type of optimal designs under a nondifferentiable criterion, namely, a design that maximizes the minimum of some measure of goodness in an experiment. Duarte and Wong (2014b) applied SIP to find minimax optimal designs for nonlinear models. Such mathematical programming tools have long been widely used in the engineering field for various optimization purposes, and it is a curiosity why such methods are not as popular in statistics as other optimization tools.

Another class of algorithms with a long history (explained in more detail in Section 21.5.1) for finding optimal designs is the class of multiplicative algorithms. Early work in this area was initiated by Titterington (1976, 1978). Recent theoretical advances and new ways to increase the speed of such algorithms have resulted in its renewed interest. In particular, Mandal and Torsney (2006) applied multiplicative algorithms to clusters of design points for better efficiency. Harman and Pronzato (2007) proposed methods to exclude nonoptimal design points so as to reduce the dimension of the problem. Dette et al. (2008) modified the multiplicative algorithm to take larger steps at each iteration but still maintain monotonic convergence. Yu (2011) combined the multiplicative algorithm, a Fedorov exchange algorithm, and a nearest neighbor exchange (NNE) strategy to form the cocktail algorithm, which appears to be much faster without sacrificing monotone convergence; see Section 21.5.3 for further discussion. See also Torsney and Martin-Martin (2009) who used multiplicative algorithms to search for optimal designs when responses are correlated. Harman (2014) proposed easy-to-implement multiplicative methods for computing D-optimal stratified designs. Here, *stratified* refers to allocating given proportions of trials to selected nonoverlapping partitions of the design space. Harman (2014) proposed two methods: one using a renormalization heuristic and the other using a barycentric algorithm.

Recently, Yang, Biedermann and Tang (2013) proposed a new iterative algorithm for computing approximate designs. At each iteration, a Newton-type algorithm is used to find the optimal weights given the current set of support points; a new support point is also added as in Fedorov's exchange method to ensure that true support points are not accidentally omitted. The method seems very promising (the authors reported computational speeds even better than the cocktail algorithm). Although the Newton steps make the implementation more involved, their algorithm readily applies to various models and design criteria.

The next few subsections describe a couple of special algorithms that seem to have garnered increased interest of late. We provide some details, including their relationship to exchange algorithms, and also demonstrate how to use them for a few specific applications in the pharmaceutical arena.

21.5.1 Multiplicative Algorithms

Multiplicative algorithms are a class of simple procedures proposed by Titterington (1976, 1978) to find approximate designs in a discrete design space; see also Silvey et al. (1978). Unlike Fedorov–Wynn type of algorithms, a multiplicative algorithm at each iteration adjusts the whole vector $\boldsymbol{w} = (w_1, \ldots, w_n)$ of design weights. Note that weights are computed for all points in the design space. Each weight is adjusted by a multiplicative factor so that relatively more weight is placed on design points whose increased weight may result in a larger gain in the objective function. Mathematically, suppose $\psi(\boldsymbol{w})$ is the objective function (design criterion evaluated at the Fisher information matrix corresponding to the weight allocation ($\boldsymbol{w}$) and then each iteration of a general multiplicative algorithm can be written as

$$w_i^{(t+1)} \propto w_i^{(t)} \left(\frac{\partial \psi(\boldsymbol{w}^{(t)})}{\partial w_i} \right)^{\rho}, \quad i = 1, \ldots, n, \tag{21.6}$$

where $\rho > 0$ and the ρth power of the derivative serves as the multiplicative factor (other functions are also possible).

This simple and general algorithm has received considerable attention; see, for example, Titterington (1976, 1978), Silvey et al. (1978), Pázman (1986), Fellman (1989), Pukelsheim and Torsney (1991), Mandal and Torsney (2006), Harman and Pronzato (2007), Dette et al. (2008), Yu (2010), and Yu (2011). On the theoretical side, Yu (2010) derived general conditions under which the multiplicative algorithm monotonically increases the objective function, which yields stable convergence. One main advantage of multiplicative algorithms is their simplicity, as illustrated by the following pseudocode.

Pseudocode for multiplicative algorithm ($\rho > 0$):

```
1. Choose starting weights $\boldsymbol{w}^{(0)} = (w_1^{(0)}, w_2^{(0)}, \ldots, w_n^{(0)})$ such that $w_j^{(0)} > 0$ for all $1 \leq j \leq n$.
2. At iteration $t$, suppose the current weights are $\boldsymbol{w}^{(t)} = (w_1^{(t)}, w_2^{(t)}, \ldots, w_n^{(t)})$. Compute $\chi_j = \partial \psi(\boldsymbol{w}) / \partial w_j$ at $\boldsymbol{w} = \boldsymbol{w}^{(t)}$, and form $w_j^{(t+1)} = w_j^{(t)} \chi_j^{\rho} / \sum_i w_i^{(t)} \chi_i^{\rho}$, $1 \leq j \leq n$.
3. Iterate steps 1 and 2 until convergence.
```

Despite their simplicity and (in certain cases) monotonic convergence, multiplicative algorithms are often slow. In Section 21.5.3, we discuss some improvements and alternatives. Some recent multiplicative algorithms are based on the Fedorov–Wynn algorithm and its modifications.

21.5.2 Application of Multiplicative Algorithm in Pharmacology

Let us consider the following compartmental model that is commonly used in pharmacokinetics. The mean of the response variable at time x is modeled as

$$\eta(x, \theta) = \theta_3[\exp(-\theta_2 x) - \exp(-\theta_1 x)], \quad x \geq 0. \tag{21.7}$$

Frequently, the mean response measures the movement of the drug concentration in the target compartment in the body, say, in the liver. Common interests in such studies are to estimate the time the drug spends inside the compartment, the peak concentration in the compartment, and the time it takes for the compartment to receive the maximum concentration. Our interest here is to select the optimal number of time points and the optimal number of subjects from which responses will be taken at each of these time points.

Since this is a nonlinear model, the optimal design will depend on the parameter $\theta = (\theta_1, \theta_2, \theta_3)'$. We compute the locally D-optimal design at the prior guess $\theta = (4.29, 0.0589, 21.80)'$ (see Atkinson et al. 2007, Example 17.4). The design space is the interval $[0, 20)$ (in minutes) discretized, specifically $x \in \{x_1, x_2, \ldots, x_n\}$ where $n = 200$ and $x_j = (j-1)/10$. The D-optimality criterion corresponds to

$$\psi_D(\boldsymbol{w}) = \log\det \boldsymbol{M}(\boldsymbol{w}), \quad \boldsymbol{M}(\boldsymbol{w}) = \sum_{i=1}^{n} w_i f(x_i, \theta) f'(x_i, \theta), \tag{21.8}$$

where the gradient vector $f'(x, \theta) \equiv \nabla_\theta \eta(x, \theta)$, often called the parameter sensitivity, is of length $m = 3$. We note that here the information matrix depends only on the weights since the whole space has been discretized, and the design problem reduces to just finding the optimal weights at these points. The multiplicative algorithm (21.6) (using $\rho = 1$, a common choice that has a convergence guarantee for the D criterion) takes a particularly simple form:

$$w_i^{(t+1)} = w_i^{(t)} m^{-1} f'(x_i, \theta) \boldsymbol{M}^{-1}\left(\boldsymbol{w}^{(t)}\right) f(x_i, \theta), \quad i = 1, \ldots, n.$$

Table 21.1 displays the design points and their weights after 2,000, 10,000, and 50,000 iterations of this algorithm. Design weights less than 0.01 are omitted. Here, the optimal design should have weights 1/3 at each of $x = 0.2, 1.4$ and 18.4. As one can readily see, the convergence of the algorithm is slow. After a large number of iterations, there is still a noticeable clustering of design weights around $x = 18.4$.

TABLE 21.1
Multiplicative Algorithm for D-Optimal Design for a Compartmental Model

x	**0.2**	**1.3**	**1.4**	**18.0**	**18.1**	**18.2**	**18.3**	**18.4**	**18.5**	**18.6**	**18.7**	**18.8**	**18.9**
$w^{(2,000)}$	0.33	0.01	0.32	0.02	0.03	0.04	0.04	0.05	0.04	0.04	0.03	0.02	0.01
$w^{(10,000)}$	0.33		0.33			0.03	0.07	0.10	0.08	0.03			
$w^{(50,000)}$	0.33		0.33				0.04	0.23	0.07				

21.5.3 Cocktail Algorithm

Several exchange strategies exist for finding approximate designs, similar to the case of exact designs. To describe these exchange strategies, let us define the directional derivative $d(i, \boldsymbol{w}) = \partial\psi((1-\delta)\boldsymbol{w} + \delta \boldsymbol{e}_i)/\partial\delta|_{\delta=0+}$ where $\boldsymbol{e}_i$ is the vector that assigns all the mass to the ith design point, $i = 1, \ldots, n$, and zero to all other design points. As before, ψ is the objective function, and $\boldsymbol{w}$ is the vector of design weights. We use the vertex direction method (VDM) (Fedorov, 1972), which is a simple iterative method: given the current $\boldsymbol{w}^{(t)}$, we first select the index $i_{\max}$ with the maximum directional derivative, that is, $d\left(i_{\max}, \boldsymbol{w}^{(t)}\right) = \max_{1 \le i \le n} d\left(i, \boldsymbol{w}^{(t)}\right)$, and then set the next iteration $\boldsymbol{w}^{(t+1)}$ as the maximizer of $\psi(\boldsymbol{w})$ along the line segment $\boldsymbol{w} = (1-\delta)\boldsymbol{w}^{(t)} + \delta \boldsymbol{e}_{i_{\max}}$, $\delta \in [0, 1]$. This one-dimensional maximization is usually not too difficult and can often be done in closed form. Plainly, each iteration of VDM moves the vector of weights $\boldsymbol{w}^{(t)}$ toward the vertex of maximum directional derivative.

Similar to VDM, one can define a general exchange between two design points i and j as follows. Given the current $\boldsymbol{w}^{(t)}$, set the new vector $\boldsymbol{w}^{(t+1)}$ as the maximizer of $\psi(\boldsymbol{w})$ along the line segment $\boldsymbol{w} = \boldsymbol{w}^{(t)} + \delta(\boldsymbol{e}_j - \boldsymbol{e}_i)$, $\delta \in [-w_j, w_i]$. In other words, with the weights at other design points unchanged, those between i and j are allocated so that the objective function is maximized. Such exchanges can set the mass at a design point at zero in one iteration, unlike the multiplicative algorithm. For example, if $\delta = w_i$, then all the mass on i is transferred to j, and $w_i^{(t+1)} = 0$. Exchanging between two design points forms the basis of Bohning's (1986) *vertex exchange method* (VEM), which uses a special choice of i and j at each iteration. Specifically, at iteration t, VEM chooses $i_{\min}$ and $i_{\max}$ such that

$$d\left(i_{\min}, \boldsymbol{w}^{(t)}\right) = \min\left\{d\left(i, \boldsymbol{w}^{(t)}\right) : \; w_i^{(t)} > 0\right\};$$

$$d\left(i_{\max}, \boldsymbol{w}^{(t)}\right) = \max\left\{d\left(i, \boldsymbol{w}^{(t)}\right) : \; 1 \le i \le n\right\}.$$

In other words, we exchange mass between a design point with maximum directional derivative and another point (among those that have positive mass) with minimum directional derivative.

Another strategy, known as the NNE, is critical to the cocktail algorithm of Yu (2011). At each iteration of NNE, we first determine an ordering on the set of support points (design points with positive mass). An ideal ordering should place similar design points close to each other. For example, suppose the design variable is a discretization of the interval $[0, 1]$, then an obvious choice is the natural ordering of the real numbers. (Note that design points with zero mass are excluded.) In general, there is no definite rule for choosing an ordering. In the context of D-optimal designs for regression problems, Yu (2011) advocates dynamically choosing an ordering at each iteration based on the L_1 distances between vectors of explanatory variables that the design points represent. Once an ordering is chosen, one performs pairwise exchanges between consecutive points on the list. For example, if the design space has four points $\{x_1, x_2, x_3, x_4\}$ and the current ordering is (x_3, x_1, x_4) (design point x_2 has zero mass and is excluded), then we exchange mass between x_3 and x_1 and then between x_1 and x_4. Each exchange involves a one-dimensional maximization that improves the design.

The cocktail algorithm of Yu (2011) builds on both the multiplicative algorithm and exchange strategies. Each iteration of the cocktail algorithm is a simple concatenation of one iteration of VDM, one iteration of the multiplicative algorithm, and the full sequence of NNE described earlier. Heuristically, NNE uses local exchanges that might complement the global nature of the multiplicative algorithm. NNE can also quickly eliminate design points (i.e., setting their masses to zero) because of the many pairwise exchanges involved. The VDM step ensures that a design point eliminated by NNE has a chance of *ressurrection* if indeed it should receive positive mass in the optimal solution; this enables theoretical proofs of convergence (to the global optimum). Empirical comparisons of the cocktail algorithm with VEM as well as several off-the-shelf optimization methods (e.g., Nelder–Mead, conjugate gradient, and quasi-Newton) show that the cocktail algorithm improves upon traditional algorithms considerably in terms of computational speed, using effectively the same stopping criterion.

21.5.4 Application of a Cocktail Algorithm in Pharmacology

As an example, let us consider the compartmental model (21.7) again. In addition to D-optimality, we also consider c-optimality, which aims to minimize the variance of an estimated scalar function of the vector of model parameters θ, say, $g(\theta)$. The optimality criterion can be written as

$$\psi_c(\boldsymbol{w}) = \boldsymbol{c}'\boldsymbol{M}^{-1}(\boldsymbol{w})\boldsymbol{c},$$

where $\boldsymbol{M}$ is as in (21.8) and $\boldsymbol{c}' \equiv \nabla g(\theta)$. Because c-optimal designs often result in singular $\boldsymbol{M}$ matrices, a small positive quantity is added to the diagonals of $\boldsymbol{M}(\boldsymbol{w})$ to stabilize the matrix inversion mentioned earlier. We emphasize that this is introduced merely to avoid numerical difficulties and does not correspond to what is intended by the c criterion. Two functions of interest (see Atkinson et al. 2007, Example 17.4) are the area under the curve (AUC), which is defined as $g_1(\theta) = \theta_3/\theta_2 - \theta_3/\theta_1$, and the time to maximum concentration (TMC), which is defined as $g_2(\theta) = (\log\theta_1 - \log\theta_2)/(\theta_1 - \theta_2)$. Table 21.2 displays the D-optimal design as well as the c-optimal designs for g_1 and g_2 found by the cocktail algorithm. These generally agree with Table 17.1 of Atkinson et al. (2007); slight discrepancies exist because we discretize the design space. In this example, it takes the cocktail algorithm only a few iterations to obtain the D-optimal design to the degree of accuracy as displayed

TABLE 21.2

Cocktail Algorithm for D- and c-Optimal Designs for a Compartmental Model

Criterion	x	Weight	Criterion Value
D-optimality	0.2	0.3333	7.3713
	1.4	0.3333	
	18.4	0.3333	
AUC	0.2	0.0137	2190.2
	17.5	0.1459	
	17.6	0.8404	
TMC	0.2	0.5916	0.028439
	3.4	0.3025	
	3.5	0.1059	

in Table 21.2. However, it takes considerably more iterations (hundreds or even thousands) to find the *c*-optimal designs.

21.6 Metaheuristic Algorithms

Recently, evolutionary algorithms have become more popular in finding optimal designs for industrial experiments. Here, we present some examples. Accelerated life testing has been used to study the degradation of reliable components of materials with high risk, such as nuclear reactors and aerospace vehicles. In order to set up a degradation testing procedure, several objectives have to be considered. For example, in such experiments, we want to have an adequate model for the degradation process to understand the relationship between degradation and failure time and, at the same time, high-quality estimates of model parameters. This leads to a highly nonlinear multiobjective optimization problem. Marseguerra et al. (2003) proposed using a GA for finding optimal designs and found nondominated solutions with two objective functions for estimating the test parameters efficiently. Similar algorithms are used for finding optimal designs in product assembly line as well. Traditional fractional factorial and response surface type of designs cannot be used when practical restrictions are imposed on factor-level combinations. Sexton et al. (2006) considered exchange algorithms and GA to compare their performances on product designs. In the two hydraulic gear pump examples and one electroacoustic transducer example they considered, the exchange algorithm performed better than the GA, although the authors noted that in the early stages of searching, GA performed better, and hence they recommended a combination of two algorithms, with GA to be used at the early stages before switching to the exchange algorithms.

GA are often used in obtaining robust parameter designs in the presence of control and noise variables, where low prediction variances for the mean response are often desirable. Goldfarb et al. (2005) used GA to obtain efficient designs for mixture processes. With the examples of soap manufacturing and another tightly constrained mixture problem, they obtained *D*-efficient designs. Rodriguez et al. (2009) used GA as well to construct optimal designs using a desirability score function (Harrington 1965) to combine dual objectives of minimizing the variances for the mean and slope into one objective function.

We focus on such algorithms inspired by nature, and so they are generally referred to as nature-inspired metaheuristic algorithms. The genesis of such algorithms could be based on a variety of observations from nature, for example, how ants colonize or how fish swim in large schools when they perceive a threat or birds fly as a flock in search of food. Generally, there is no mathematical proof that shows metaheuristic algorithms will converge to the optimum; that is why they are called metaheuristic in the first place! However, repeated experiences from researchers in many fields report that they frequently do find the optimum, and if they do not, these algorithms get to the proximity of the optimum quickly.

There are many metaheuristic algorithms in the optimization literature, and it is a curiosity why statisticians do not seem to explore and use more of them in their work. The most common examples in the statistical literature are simulated annealing (SA) and GA, discussed in the succeeding text. There are many newer ones, such as PSO, bat algorithm, differential evolution algorithm, and cuckoo algorithm. For space consideration, we discuss in the succeeding text only PSO, which is just beginning to appear in the statistical literature. All are nature-inspired algorithms with different search techniques based on metaheuristics. Yang (2010) has a brief but good discussion on the other algorithms.

21.6.1 Genetic Algorithms

GA are one of the evolutionary algorithms that are very popular for finding exact optimal designs. Developed by John Holland and his colleagues at the University of Michigan (Holland 1975), these algorithms mimic Darwin's theory of evolution. The metaphors of natural selection, crossbreeding, and mutation have been helpful in providing a framework to explain how and why GA work. In our context, this translates to the heuristic that from two "good" design points, one should sometimes be able to generate an even "better" design point. The objective function to be optimized, denoted by ψ as in (21.5), can be treated as a *black box* without the mathematical requirements of continuity, differentiability, convexity, or other properties required by many traditional algorithms. Because of its simplicity, this algorithm has also gained popularity in finding optimal designs for computer experiments (see Bates et al. 2003; Liefvendahl and Stocki 2006; Crombecq and Dhaene 2010). It is an iterative algorithm that starts with a set of candidates and can explore very large spaces of candidate solutions. Convergence is usually not guaranteed but GA often yield satisfactory results.

In an attempt to understand how GA function as optimizers, Reeves and Wright (1999) considered GA as a form of sequential experimental design. (For details of sequential design algorithms, see Chapter 19.) Recently, GA have been used quite successfully in solving statistical problems, particularly for finding near-optimal designs (Hamada et al. 2001; Heredia-Langner et al. 2003, 2004). In this chapter, we will occasionally deviate from what is used in the optimization literature and present our algorithm in the context of search for optimal designs. One of the simplest version of the algorithm process is as follows:

1. *Solution representation*: For a single design point, a factor at a particular level is called a *gene*. The factor combinations, that is, the entire design point or *run*, are called a *chromosome*. Using the notation of Section 21.4.2, $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})$ is a chromosome, whereas each x_{ij} is called a gene. Note that it is possible to define the chromosome a little differently, as we will see in the Broudiscou et al. (1996) example later. Initially a large population of random candidate solutions (design points) is generated; these are then continually transformed following steps (2) and (3).
2. *Select* the best and eliminate the worst design point on the basis of a fitness criterion (e.g., the higher the better for a maximization problem) to generate the next population of design points.
3. *Reproduce* to transform the population into another set of solutions by applying the genetic operations of *crossover* and *mutation*. Different variants of crossover and mutations are available in literature.
   a. *Crossover* : A pair of design points (chromosomes) are split at a random position, and the head of one is combined with the tail of other and vice versa.
   b. *Mutation* : The state (i.e., level) of a randomly chosen factor is changed. This helps the search avoid being trapped at local optima.
4. *Repeat* steps (2) and (3) until some convergence criterion is met (usually no significant improvement for several iterations).

Note that in this sketched algorithm, it is more in the line of solving an optimization problem than a design problem. For example, we inherently assumed that all the designs we consider consist of only one run, which is not the case for most of our problems. But the aforementioned algorithm can be modified very easily to solve the standard design problems (e.g., finding orthogonal arrays with fixed run size). Next, we illustrate the algorithm stated earlier with an optimization problem, mainly for simplicity. After that, we will discuss common design problems where the entire design with multiple runs is taken as a chromosome, and the collection of such designs represent the population.

Let us illustrate the algorithm sketched earlier with a *black box* function provided by Levy and Montalvo (1985):

$$\psi(x_{i1},\ldots,x_{ip}) = \sin^2\left\{\pi\left(\frac{x_{i1}+2}{4}\right)\right\} + \sum_{k=1}^{p-1}\left(\frac{x_{ik}-2}{4}\right)^2\left\{1+10\sin^2\left(\pi\left(\frac{x_{ik}+2}{4}\right)+1\right)\right\}$$
$$+\left(\frac{x_{ip}-2}{4}\right)^2\left\{1+\sin^2\left(2\pi\left(x_{ip}-1\right)\right)\right\}. \qquad (21.9)$$

Here, $p = 4$ and only integer values of x_{ik}'s ($0 \leq x_{ik} \leq 10$) are considered. This corresponds to an experiment with four factors each at 11 levels denoted by $0, 1, \ldots, 10$. Suppose we start with a random population of size 10 given in Table 21.3. In this case, the ten runs $x_1 = (2, 1, 4, 5)', \ldots, x_{10} = (0, 9, 8, 7)'$ are the ten chromosomes that constitute the initial population.

Suppose that the best two design points, namely, $(10, 6, 7, 8)'$ and $(0, 9, 8, 7)$, are chosen as *parents* to *reproduce* Then suppose crossover happens at location 2, so the new sets of design points will be $(10, 6, 8, 7)'$ and $(0, 9, 7, 8)'$ (the first two factors of the first design point are combined with the last two factors of the second design point and vice versa). Now suppose for $(10, 6, 8, 7)'$ the mutation happens at location 3 (for factor C) and the level is changed from 8 to 7. Then the resulting design point becomes $(10, 6, 7, 7)'$ with the value of Levy–Montalvo function as 26.81. Note that this value is worse than $(10, 6, 7, 8)'$ and that is not unexpected. What is expected is that, on average, when 10 new *offsprings* (design points) are generated in this fashion, some of them will be "good". Those new 10 offsprings

TABLE 21.3

Initial Population for Levy–Montalvo Example

A	B	C	D	ψ
2	1	4	5	1.63
3	4	1	7	3.13
1	1	8	8	11.66
10	6	7	8	26.81
9	5	5	9	5.20
7	8	2	8	11.57
2	3	5	3	1.54
6	5	4	6	2.80
5	9	5	8	8.02
0	9	8	7	15.83

will constitute the second generation of the population, and the same process continues. Usually hundreds or even thousands of generations are needed to obtain a near-optimal design. Mandal et al. (2006) used an efficient modification of GA on this same example and showed that it produced good results.

Broudiscou et al. (1996) used GA for constructing D-optimal designs, in the context of an antigen/antibody test to diagnose an infection to a virus. There were six factors with a total of $7 \times 6 \times 6 \times 5 \times 3 \times 3 = 11,340$ combinations. In this example, the basic exchange algorithms discussed in Section 21.4.2 are not very efficient. Consider a design with $n = 28$ points. In Fedorov's exchange algorithm, during each iteration, a design point $\boldsymbol{x}_i$ will be replaced by the point $\boldsymbol{x}_*$ that maximizes $\Delta(\boldsymbol{x}_i, \boldsymbol{x}_*)$ of (21.4). Also, one has to evaluate all the pairs $(\boldsymbol{x}_i, \boldsymbol{x}_*)$. Naturally, when the total number of candidate design points is high (11,340 in this case), even with moderate run size (28 in this case), the algorithm will be slow because finding the "best" combination, at each iteration, is time-consuming. Stochastic search-based algorithms, such as GA, turn out to be very efficient in such situations. The authors reported a D-optimal design in Table 5 of their paper, for a mixed-level factorial main effects model. In this example, the model matrix $\boldsymbol{X}$ has $1 + 6 + 5 \times 2 + 4 + 2 \times 2 = 25$ columns. The objective function is $\psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \det[f'(\boldsymbol{X})f(\boldsymbol{X})]$, as mentioned before, and our objective is to maximize ψ. In this case, the chromosomes are possible designs obtained by juxtaposing the rows of the design matrix, such as $(\boldsymbol{x}'_1, \boldsymbol{x}'_2, \ldots, \boldsymbol{x}'_{28})'$. Each chromosome has $6 \times 28 = 168$ genes.

Hamada et al. (2001) used GA to find near-optimal Bayesian designs for regression models. The objective function ψ in their case was the expected Shannon information gain of the posterior distribution. It is equivalent to choosing designs that maximize the expected Kullback–Leibler distance between the prior and posterior distributions (Chaloner and Verdinelli 1995). The reader is referred to the pseudocode given in the appendix of their paper. They also considered a design with p factors and n runs, and each factor level as a gene such that each chromosome (where it is the design) had np genes.

Kao et al. (2009) used GA to construct optimal designs for event-related fMRI studies. The authors considered multiple objectives, and ψ was defined to reflect all of them. In that application, the chromosomes can be 300–600 genes long, each gene taking 5–13 different values. See Chapter 25 for a detailed discussion of their approach.

Pseudocode for GA:

```
1. Generation 0: Generate $M$ random designs, evaluate them by the objective function
   $\psi$, and order them by their fitness ($\psi$-values).
2. For generations $g = 1, \ldots, G$, the following apply:
   a. With probability proportional to fitness, draw with replacement $M/2$ pairs of
      designs to crossover—select a random cut point, and exchange the correspond-
      ing fractions of genes in paired designs to generate $M$ offspring designs.
   b. Randomly select $q\%$ of the genes from the $M$ offspring designs and perform
      mutation to generate $M$ new offspring designs.
   c. Obtain the fitness scores of the new offsprings.
   d. Repeat steps (a through c) until stopping criterion is met.
```

As mentioned before, there are several variants of GA. Being one of the metaheuristic algorithms, although the performance of GA in finding optimal design is, in general, not greatly affected by the functionality of the algorithm, in order to apply this algorithm, a number of parameters need to be determined beforehand. These include the population size (M), mutation rates (q), crossover, and mutation locations. Such details of implementation, however, are essential for anyone to reproduce the results of a search, even after accounting for the randomness of the procedure. Lin et al. (2014) explored the merits of GA in the context of design of experiments and discussed some elements that should be considered for an effective implementation.

21.6.2 Simulated Annealing

While GA mimic Darwin's theory of evolution, SA simulates the process called annealing in metallurgy where some physical properties of a material are altered by heating to above a critical temperature, followed by gradual cooling. It starts with a random design and then moves forward by replacing the current solution by another solution with the hope of finding near-optimal solutions. Introduced by Kirkpatrick et al. (1983), SA is a probabilistic global optimization technique and has two basic features. The first one is motivated by the Metropolis algorithm (Metropolis et al. 1953), in which designs that are worse than the current one are sometimes accepted in order to better explore the design space. The other one is the strategy for lowering the *temperature* (T), by which the probability of inclusion of new "bad" solutions is controlled. In the traditional formulation for a minimization problem, initially this time-varying parameter T is set at a high value in order to explore the search space as much as possible. At each step, the value of the objective function is calculated and compared with the best design at hand. In the context of optimal design search, designs with higher values of the objective function will be accepted according to the Metropolis algorithm if $e^{-\Delta D/T} > U$ where ΔD is the change of the objective function, T is the *current temperature*, and U is a uniform random number. Usually, the temperature is lowered at some specified intervals at a geometric rate. The higher the temperature value, the more unstable the system is, and the more frequently "worse" designs are accepted. This helps explore the entire design space and jump out of local optimum values of the objective function. As the process continues, the temperature decreases and leads to a steady state. The temperature T, however, should not be decreased too quickly in order to avoid getting trapped at local optima. There are various *annealing schedules* for lowering the temperature. Similar to GA, the algorithm also usually stops when no significant improvements are observed for several iterations.

Woods (2010) used SA to obtain optimal designs when the outcome is a binary variable. The objective of the study is to maximize $\psi(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ the log determinant of the Fisher information matrix that depends on the unknown values of the parameters. As in Section 21.4.2, here, $\boldsymbol{x}_i$ represents a vector $(x_{i1}, \ldots, x_{ip})$ where x_{ij} represents the value of the jth variable in the ith trial, with the constraint that $-1 \leq x_{ij} \leq 1$ ($j = 1, \ldots, p$; $i = 1, \ldots, n$). In this case, x_{ij} has been perturbed to a new design point x_{ij}^{new} by

$$x_{ij}^{new} = \min\{1, \max[-1, x_{ij} + ud_T]\}$$

where u is a random number drawn from a uniform $U[-1, 1]$ distribution and d_T is the size of the maximum allowed perturbation at temperature T. The new (perturbed) design is compared with the original design via the calculation of the objective function ψ, and the new design is accepted if its ψ value is greater than that of the original one. Otherwise, the new design is accepted with probability $\min\left\{1, \exp\left(\frac{\psi(\xi^{new}) - \psi(\xi^{original})}{T}\right)\right\}$. Here, $\xi^{original}$ and ξ^{new} are, respectively, the n-run designs supported at the original and new sets of points $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. Both the temperature T and perturbation size d_T are decreased geometrically.

Bohachevsky et al. (1986) developed a generalized SA method for function optimization and noted that this algorithm has "the ability to migrate through a sequence of local extrema in search of the global solution and to recognize when the global extremum has been located." Meyer and Nachtsheim (1988) used SA (programmed in ANSI-77 standard FORTRAN) for constructing exact D-optimal designs. They evaluated their methods for both finite and continuous design spaces. Fang and Wiens (2000) used SA for obtaining integer-valued designs. In their paper, they mentioned that, following Haines (1987), they initially chose T in such a way that the acceptance rate is at least 50%. As discussed by Press et al. (1992), they decreased T by a factor of 0.9 after every 100 iterations. Among others, Zhou (2008) and Wilmut and Zhou (2011) used SA for obtaining D-optimal minimax designs.

21.6.3 Particle Swarm Optimization

PSO was proposed about 18 years ago by Kennedy and Eberhart (1995), and it has slowly but surely gained a lot of attention in the past 10 years. Researchers continue to report their successes with the algorithm for solving large-scale, complex optimization problems in several fields, including finance, engineering, biosciences, monitoring power systems, social networking, and behavioral patterns. The rate of success and excitement generated by PSO has led to at least one annual conference solely devoted to PSO for more than a decade and usually sponsored by IEEE. There are several websites that provide in-depth discussions of PSO with codes, tutorials, and both real and illustrative applications. An example of such a website is http://www.swarmintelligence.org/index.php. Currently, there are at least three journals devoted to publishing research and applications based on swarm intelligence PSO methods.

The special features of the PSO techniques are that the method itself is remarkably simple to implement and flexible, requires no assumption on the function to be optimized, and requires specification of only a few easy-to-use tuning parameters to values that work well for a large class of problems. This is unlike other algorithms such as the genetic or SA algorithms that can be sensitive to the choice of tuning parameters. For PSO, typically only two parameters seem to matter—the number of particles and the number of iterations—with the rest taking on the default values. A larger number of particles generate more starting designs that more likely cover the search space more adequately and so usually produces a higher quality solution. A probable downside is that it may take longer time to arrive at the optimum because more communication time is required by the larger number of particles to decide where the global optimum is. The user also has to specify the maximum number of iterations allowed for the algorithm, but this usually is inconsequential for small dimensional optimization problems. This is because the search time is typically very short, and so one can try repeatedly and find the optimal design quickly for most problems. Each particle is a potential solution of the optimization problem, and at every iteration, each has a fitness value determined by the design criterion.

There are two key equations in the PSO algorithm that define its search to optimize a user-selected objective function ψ. At the t and $t+1$ iterations, the movement of particle i is governed by

$$\boldsymbol{v}_i^{t+1} = \omega_t \boldsymbol{v}_i^t + \gamma_1 \boldsymbol{\beta}_1 \odot (\boldsymbol{p}_i - \boldsymbol{x}_i^t) + \gamma_2 \boldsymbol{\beta}_2 \odot (\boldsymbol{p}_g - \boldsymbol{x}_i^t) \qquad (21.10)$$

and

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}. \tag{21.11}$$

Here $\boldsymbol{v}_i^t$ is the particle velocity at time t and $\boldsymbol{x}_i^t$ is the current particle position at time t. The inertia weight ω_t adjusts the influence of the former velocity and can be a constant or a decreasing function with values between 0 and 1. For example, Eberhart and Shi (2000) used a linearly decreasing function over the specified time range with initial value 0.9 and end value 0.4. Further, the vector $\boldsymbol{p}_i$ is the personal best (optimal) position as encountered by the ith particle, and the vector $\boldsymbol{p}_g$ is the global best (optimal) position as encountered by all particles, up to time t. This means that up to time t, the personal best for particle i is $pbest_i = \psi(\boldsymbol{p}_i)$ and, for all particles, $gbest = \psi(\boldsymbol{p}_g)$ is the optimal value. The two random vectors in the PSO algorithm are $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$, and their components are usually taken to be independent random variables from $U(0, 1)$. The constant γ_1 is the cognitive learning factor, and γ_2 is the social learning factor. These two constants determine how each particle moves toward its own personal best position and the overall global best position. The default values for these two constants in the PSO codes are $\gamma_1 = \gamma_2 = 2$, and they seem to work well in practice for nearly all problems that we have investigated so far. Note that in (21.10), the product in the last two terms is the Hadamard product. The pseudocode for the PSO procedure using q particles is given in the succeeding text.

Pseudocode for PSO algorithm is as follows:

```
1. Initialize particles.
   a. Initiate positions $x_i$ and velocities $v_i$ for $i = 1, \ldots, q$.
   b. Calculate the fitness values $\psi(x_i)$ for $i = 1, \ldots, q$.
   c. Initialize the personal best positions $p_i = x_i$ and the global best position $p_g$.
2. Repeat until stopping criteria are satisfied.
   a. Calculate particle velocity according to Equation (21.10).
   b. Update particle position according to Equation (21.11).
   c. Update the fitness values $\psi(x_i)$.
   d. Update personal and global best positions $p_i$ and $p_g$.
3. Output $p_g = \arg\min \psi(p_i)$ with $gbest = \psi(p_g)$.
```

We have set up mirror websites at http://optimal-design.biostat.ucla.edu/podpack/, http://www.math.ntu.edu.tw/~optdesign/, and http://www.stat.ncku.edu.tw/optdesign, where the reader can download our PSO P-codes, run them, and verify some of the results in this chapter. Many of the PSO codes can be readily changed to find another type of optimal design for the same model or for a different model. Typically, the only changes that are required are in the information matrix and the design criterion.

The sites are new and are still under construction as improvements are made. We alert the reader that some of the notation on these sites may be different from that used in this chapter. The sites have instructions for downloading MATLAB® P-codes and running the codes for finding various types of optimal designs. On the interface window, we provide default values for two PSO parameters that had successfully found the optimal design before, the number of particles, and the number of iterations; all other parameters are default values recommended by PSO and are not displayed. The interface window also provides the swarm plot showing how the swarm of initial candidate designs converges or not to the optimal location, along with a plot of the directional derivative of the design criterion for the PSO-generated design to confirm its optimality or not. The ease of use and flexibility of PSO are compelling compared with current methods of finding optimal designs.

As examples, we refer the reader to the download webpage on one of the aforementioned websites, where under the heading Part A, we have codes for finding minimax optimal designs. The first example concerns finding a design to minimize the maximum variance of the fitted response across the design space when errors have a known heteroscedastic structure, and the second example concerns E-optimality where we seek a design that minimizes the maximum eigenvalue of the inverse of the information matrix (Ehrenfeld, 1955). We invite the reader to try out the PSO codes on the website for generating various types of optimal designs for the compartmental model discussed earlier in Section 21.5.1 and compare results in Tables 21.1 and 21.2. Other sample applications of using PSO to design real studies available from the website include finding different locally optimal designs for the simple and quadratic logistic models, D-optimal designs for mixture models, locally D-optimal designs for the four-parameter Hill model used in education and biomedical studies, locally D-optimal designs for an exponential survival model with type I right censoring, and locally D-optimal designs for a double-exponential model used in monitoring tumor regrowth rate.

PSO techniques seem like a very under utilized tool in statistics to date. They seem ideally suited for finding optimal experimental designs. This is because many applications having a design close to the optimum (without knowing the optimum) may suffice for most practical purposes. When we work with approximate designs, the convexity assumption in the design criterion implies that the skeptic can also always check the quality of the PSO-generated design using an equivalence theorem. Intuitively, PSO is attractive because it uses many starting designs (particles) at the initial stage to cover the search space, and so one can expect such an approach is preferable to methods that use only a single starting design.

21.7 Summary

This chapter discusses algorithmic searches for an optimal design for a statistical model described in (21.2). We reviewed a few algorithms commonly used to find an optimal design for the problem and also newer algorithms, such as particle swarm–based algorithms.

Our discussion has focused on a single-objective study. In practice, experiments may have more than one goal or objective, and the implemented design should carefully incorporate all the objectives at the onset. In the past, the practice was to design the study to satisfy the most important objective and hope that the same design also does well in terms of other objectives in the study. Nowadays, multi objective optimization problems can be handled by constructing a multiple-objective optimal design that directly accommodates all experimental goals at the same time. These techniques require that the objectives be first prioritized in order of their importance. By construction, the multiple-objective optimal design captures the varying degrees of importance of the various objectives and delivers user-specified efficiencies for the objectives according to their importance; see details in Cook and Wong (1994), for example. In particular, it can be shown that many of the algorithms for finding a single-objective optimal design can be directly applied to find multiple-objective optimal designs as well (Cook and Wong 1994; Wong 1999). Chapter 25 constructs some specific multiple-objective optimal designs for event-related fMRI studies.

We have not made a clear distinction between finding optimal exact or approximate designs but note that some algorithms are more flexible than others. There is no algorithm that is universally best for solving all optimization problems. Each algorithm, by construction, has its own unique strengths, weaknesses, and restrictions. For example, both multiplicative algorithms and SDP-based methods require that the search space be discretized, but PSO can work well either in a continuous or a discrete search space. Further, the performance of some algorithms, such as the GA, can be highly dependent on input values of the tuning parameters, while others, such as PSO-based algorithms, are less so. It is therefore important to fully appreciate the properties of the algorithm before implementing it to find the optimal design for the problem at hand. Frequently, for more complicated optimization problems, a hybrid algorithm that combines two or more different algorithms can prove effective because it incorporates the unique strengths from the component algorithms.

We would be remiss not to mention that there are canned software packages in commercial statistical software for finding optimal designs. For example, Atkinson et al. (2007) provided SAS codes for finding a variety of optimal designs based on various types of algorithms. Currently, algorithms are available for generating D-, c-, A-optimal designs and optimal designs found under differentiable criteria. Other statistical packages also focus on such types of optimal designs. Minimax optimal designs are notoriously difficult to find, and we are not aware of any commercial package that handles them. GA and PSO that do not require the objective function to be differentiable would seem more appropriate for such problems. Examples of work in this area are Zhou (2008) and, most recently, Chen et al. (2014) and Qiu et al. (2014).

In summary, algorithms are crucial for finding optimal designs to solve real problems and will be more important as scientists increasingly use more realistic models to reflect the complexities of the underlying process. Consequently, analytical descriptions of optimal designs will be more difficult and most likely impossible to obtain. Numerical methods are therefore necessary, and more effective algorithms should be developed and applied to solve real-world design problems. It therefore behooves the design community to always keep a constant eye of current and new optimization techniques used in the optimization literature and investigate their suitability and efficiency for finding optimal designs for a statistical problem.

Acknowledgments

The research of Dr. Mandal was in part supported by NSF grant DMS-09-05731 and NSA grant H98230-13-1-025. Dr. Wong is especially thankful to his collaborators in Taiwan for introducing him to PSO and their continuing research in this exciting topic. They are Professor Weichung Wang from the Department of Mathematics at National Taiwan University and Professor Ray-Bing Chen from the Department of Statistics at National Cheng Kung University. Thanks also go to several students from National Taiwan

University and UCLA, who all wrote and implemented MATLAB codes to find various types of optimal designs using PSO on the website. 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 under Award Number R01GM107639. The content is solely the responsibility of the author and does not necessarily represent the official views of the National Institutes of Health. Dr. Yu would like to thank Professor Holger Dette for stimulating discussions concerning algorithms as well as minimax designs.

All authors are grateful to the editorial team for taking a lot of time to carefully read earlier versions of this chapter and providing excellent feedback. We thank them very much for their guidance throughout our preparation of this chapter. We also thank Guanghao Qi from the Department of Mathematics at Fudan University for comments on this chapter.

References

Atashgah, A. B. and A. Seifi, Optimal design of multi-response experiments using semi-definite programming, *Optimization in Engineering*, 10, 75–90, 2009.

Atkinson, A. C., A. N. Donev, and R. D. Tobias, *Optimum Experimental Designs, with SAS*, Oxford University Press, Oxford, U.K. 2007.

Bates, S. J., J. Sienze, and D. S. Langley, Formulation of the AudzeEglais uniform Latin hypercube design of experiments, *Advances in Engineering Software*, 34, 493–506, 2003.

Berger, M. P. F., J. King, and W. K. Wong, Minimax designs for item response theory models, *Psychometrika*, 65, 377–390, 2000.

Bohachevsky, I. O., M. E. Johnson, and M. L. Stein, Generalized simulated annealing for function optimization, *Technometrics*, 28, 209–217, 1986.

Böhning, D., A vertex-exchange-method in *D*-optimal design theory, *Metrika*, 33, 337–347, 1986.

Broudiscou, A., R. Leardi, and R. Phan-Tan-Luu, Genetic algorithm as a tool for selection of *D*-optimal design, *Chemometrics and Intelligent Laboratory Systems*, 35, 105–116, 1996.

Brown, L. D. and W. K. Wong, An algorithmic construction of optimal minimax designs for heteroscedastic linear models, *Journal of Statistical Planning and Inference*, 85, 103–114, 2000.

Chaloner, K. and K. Larntz, Optimal Bayesian design applied to logistic regression experiments, *Journal of Statistical Planning and Inference*, 21, 191–208, 1989.

Chaloner, K. and I. Verdinelli, Bayesian experimental design: A review, *Statistical Science*, 10, 273–304, 1995.

Chang, S., An algorithm to generate near *D*-optimal designs for multiple response surface models, *IIE Transactions*, 29, 1073–1081, 1997.

Chen, R. B., S. P. Chang, W. Wang, H. C. Tung, and W. K. Wong, Minimax optimal designs via particle swarm optimization methods, *Statistics and Computing*, 2014. doi:10.1007/s11222-014-9466-0.

Chen, R. B., W. K. Wong, and K. Y. Li, Optimal minimax designs for estimating response surface over a prespecified region in a heteroscedastic model, *Statistics and Probability Letters*, 78, 1914–1921, 2008.

Cook, R. D. and C. J. Nachtsheim, A comparison of algorithms for constructing exact *D*-optimal designs, *Technometrics*, 22(3), 315–324, 1980.

Cook, R. D. and C. J. Nachtsheim, Model robust, linear-optimal designs, *Technometrics*, 24(1), 49–54, 1982.

Cook, R. D. and W. K. Wong, On the equivalence of constrained and compound optimal designs, *Journal of American Statistical Association*, 89, 687–692, 1994.

Crombecq, K. and T. Dhaene, Generating sequential space-filling designs using genetic algorithms and Monte Carlo methods, *Lecture Notes in Computer Science*, Vol. 6457. Springer-Verlag: Berlin, Germany, pp. 80–88, 2010.

Dean, A. M. and S. M. Lewis (eds.) Screening: Methods for Experimentation in Industry, Drug Discovery and Genetics, Springer-Verlag: New York, 2006.

Dette, H., A. Pepelyshev, and A. Zhigljavsky, Improving updating rules in multiplicative algorithms for computing D-optimal designs, *Computational Statistics & Data Analysis*, 53, 312–320, 2008.

Dette, H. and W. J. Studden, *The Theory of Canonical Moments with Applications in Statistics, Probability, and Analysis*, Wiley: New York, 1997.

Duarte, B. P. M. and W. K. Wong, Finding Bayesian optimal designs for nonlinear models: A semidefinite programming based approach, *International Statistical Review*. 2014a. doi:10.1111/insr.12073.

Duarte, B. P. M. and W. K. Wong, A semi-infinite programming based algorithm for finding minimax D-optimal designs for nonlinear models, *Statistics and Computing*, 24, 1063–1080, 2014b.

Eberhart, R. C. and Y. Shi, Comparing inertia weights and constriction factors in particle swarm optimization, *Proceedings of the 2000 Congress on Evolutionary Computation*, 1, 84–88, 2000.

Ehrenfeld, S., On the efficiencies of experimental designs, *Annals of Mathematical Statistics*, 26, 247–255, 1955.

Fang, Z. and D. P. Wiens, Integer-valued, minimax robust designs for estimation and extrapolation in heteroscedastic, approximately linear models, *Journal of American Statistical Association*, 95, 807–818, 2000.

Fedorov, V. V., Theory of optimal experiments, Preprint 7 LSM, Izd-vo Moscow State University, Moscow, Russia, 1969.

Fedorov, V. V., *Theory of Optimal Experiments*, translated by W. J. Studden and E. M. Klimko, Academic Press: New York, 1972.

Fellman, J., An empirical study of a class of iterative searches for optimal designs, *Journal of Statistical Planning and Inference*, 21, 85–92, 1989.

Filová, L., M. Trnovská, and R. Harman, Computing maximin efficient experimental designs using the methods of semidefinite programming, *Metrika*, 75, 709–719, 2012.

Goldfarb, H. B., C. M. Borror, D. C. Montgomery, and C. M. Anderson-Cook, Using genetic algorithms to generate mixture-process experimental designs involving control and noise variables, *Journal of Quality Technology*, 37, 60–74, 2005.

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

Hamada, M., H. D. Martz, C. S. Reese, and A. G. Wilson, Finding near-optimal Bayesian designs via genetic algorithms, *The American Statistician*, 55, 175–181, 2001.

Harman, R., Multiplicative methods for computing D-optimal stratified designs of experiments, *Journal of Statistical Planning and Inference*, 146, 82–94, 2014.

Harman, R. and L. Pronzato, Improvements on removing nonoptimal support points in D-optimum design algorithms, *Statistics and Probability Letters*, 77, 90–94, 2007.

Harrington, E. C. Jr., The desirability function, *Industrial Quality Control*, 21, 494–498, 1965.

Heredia-Langner, A., W. M. Carlyle, D. C. Montgomery, C. M. Borror, and G. C. Runger, Genetic algorithms for the construction of D-optimal designs, *Journal of Quality Technology*, 35, 28–46, 2003.

Heredia-Langner, A., D. C. Montgomery, W. M. Carlyle, and C. M. Borror, Model-robust optimal designs: A genetic algorithm approach, *Journal of Quality Technology*, 36 (3), 263–279, 2004.

Holland, J. M. *Adaptation in Natural and Artificial Systems*, University of Michigan Press: Ann Arbor, MI, 1975.

Johnson, M. E. and C. J. Nachtsheim, Some guidelines for constructing exact D-optimal designs on convex design spaces, *Technometrics*, 25(3), 271–277, 1983.

Kao, M. H., A. Mandal, N. Lazar, and J. Stufken, Multi-objective optimal experimental designs for event-related fMRI studies, *NeuroImage*, 44, 849–856, 2009.

Kennedy, J. and R. Eberhart, Particle swarm optimization, *Proceedings of IEEE International Conference on Neural Networks, IV*, Perth, Australia, pp. 1942–1948, 1995.

Kiefer, J., General equivalence theory for optimum designs (approximate theory), *Annals of Statistics*, 2, 849–879, 1974.

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

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi, Optimization by simulated annealing, *Science,* New series, 220, 4598, 671–680, 1983.

Levy, A. V. and A. Montalvo, The tunnelling algorithm for the global minimization of functions, *SIAM Journal of Scientific and Statistical Computing*, 6, 15–29, 1985.

Liefvendahl, M. and A. Stocki, A study on algorithms for optimization of Latin hypercubes, *Journal of Statistical Planning and Inference*, 136, 3231–3247, 2006.

Lin, C. D., C. M. Anderson-Cook, M. S. Hamada, L. M. Moore, and R. R. Sitter, Using genetic algorithms to design experiments: A review, *Quality and Reliability Engineering International*, 31, 155–167, 2015.

Mandal, A., Johnson, K., Wu, C. F. J., and Bornmeier, D., Identifying promising compounds in drug discovery: Genetic algorithms and some new statistical techniques, *Journal of Chemical Information and Modeling*, 47, 981–988, 2007.

Mandal, A., P. Ranjan, and C. F. J. Wu, $\mathcal{G}$-SELC: Optimization by sequential elimination of level combinations using genetic algorithms and Gaussian processes, *Annals of Applied Statistics*, 3, 398–421, 2009.

Mandal, A., C. F. J. Wu, and K. Johnson, SELC: Sequential elimination of level combinations by means of modified genetic algorithms, *Technometrics*, 48, 273–283, 2006.

Mandal, S. and B. Torsney, Construction of optimal designs using a clustering approach, *Journal of Statistical Planning and Inference*, 136, 1120–1134, 2006.

Marseguerra, M., E. Zio, and M. Cipollone, Designing optimal degradation tests via multi-objective genetic algorithms, *Reliability Engineering and System Safety*, 79, 87–94, 2003.

Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, Equation of state calculations by fast computing machines, *Journal of Chemical Physics*, 21, 1087–1092, 1953.

Meyer, R. K. and C. J. Nachtsheim, Constructing exact D-optimal experimental designs by simulated annealing, *American Journal of Mathematical and Management Sciences*, 8, 329–359, 1988.

Meyer, R. K. and C. J. Nachtsheim, The coordinate-exchange algorithm for constructing exact optimal experimental designs, *Technometrics*, 37, 60–69, 1995.

Mitchell, T. J., An algorithm for the construction of D-optimal experimental designs, *Technometrics*, 20, 203–210, 1974.

Nguyen, N. K. and A. Miller, A review of exchange algorithms for constructing discrete D-optimal designs, *Computational Statistics and Data Analysis*, 14, 489–498, 1992.

Papp, D., Optimal designs for rational function regression, *Journal of the American Statistical Association*, 107, 400–411, 2012.

Pázman, A., Foundations of Optimum Experimental Design, Reidel: Dordrecht, the Netherlands, 1986.

Press, W. H., S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing,* 2nd ed., Cambridge University Press, Cambridge, U.K., 1992.

Pukelsheim, F. and B. Torsney, Optimal weights for experimental designs on linearly independent support points, *Annals of Statistics*, 19, 1614–1625, 1991.

Qiu, J., R. B. Chen, W. Wang, and W. K. Wong, Using animal instincts to design efficient biomedical studies via particle swarm optimization, *Swarm and Evolutionary Computation*, 18, 1–10, 2014.

Ranjan, P., D. Bingham, and G. Michailidis, Sequential experiment design for contour estimation from complex computer codes, *Technometrics*, 50 (4), 527–541, 2008.

Reemtsen, R. and J. J. Rückman, Semi-Infinite Programming, Kluwer Academic Publishers: Dordrecht, the Netherlands, 1998.

Reeves, C. L. and C. C. Wright, Genetic algorithms and the design of experiments, In Davis, L. D.; DeJong, K.; Vose, M. D., and Whitley, L. D. (eds.), *Evolutionary Algorithms: IMA Volumes in Mathematics and its Applications, Vol 111*. Springer-Verlag: New York, pp. 207–226, 1999.

Rodriguez, M., B. Jones, C. M. Borror, and D. C. Mongomery, Generating and assessing exact G-optimal designs, *Journal of Quality Technology*, 42, 3–20, 2010.

Rodriguez, M., D. C. Montgomery, and C. M. Borror, Generating experimental designs involving control and noise variables using genetic algorithms, *Quality and Reliability Engineering International*, 25, 1045–1065, 2009.

Sexton, C. J., D. K. Anthony, S. M. Lewis, C. P. Please, and A. J. Keane, Design of experiment algorithms for assembled products, *Journal of Quality Technology*, 38, 298–308, 2006.

Silvey, S. D., *Optimal Design*, Chapman & Hall: London, U.K., 1980.

Silvey, S. D., D. M. Titterington, and B. Torsney, An algorithm for optimal designs on a finite design space, *Communications in Statistics–Theory and Methods*, 14, 1379–1389, 1978.

Smith, K., On the standard deviations of adjusted and interpolated values of an observed polynomial function and its constants and the guidance they give towards a proper choice of the distribution of observations, *Biometrika*, 12, 1–85, 1918.

Syed, M. N., I. Kotsireas, and P. M. Pardalos, *D*-optimal designs: A mathematical programming approach using cyclotomic cosets, *Informatica*, 22, 577–587, 2011.

Titterington, D. M., Algorithms for computing *D*-optimal design on finite design spaces, *in Proceedings of the 1976 Conference on Information Science and Systems*, John Hopkins University: Baltimore, MD, Vol. 3, pp. 213–216, 1976.

Titterington, D. M., Estimation of correlation coefficients by ellipsoidal trimming, *Applied Statistics*, 27, 227–234, 1978.

Torsney, B. and R. Martin-Martin, Multiplicative algorithms for computing optimum designs, *Journal of Statistical Planning and Inference*, 139, 3947–3961, 2009.

Vandenberghe, L. and S. Boyd, Semidefinite programming, *SIAM Review*, 38, 49–95, 1996.

Welch, W. J., Branch and bound search for experimental designs based on *D*-optimality and other criteria, *Technometrics*, 24(1), 41–48, 1982.

Wilmut, M. and J. Zhou, *D*-optimal minimax design criterion for two-level fractional factorial designs, *Journal of Statistical Planning and Inference*, 141, 576–587, 2011.

Wong, W. K., A unified approach to the construction of mini-max Designs, *Biometrika*, 79, 611–620, 1992.

Wong, W. K. and R. D. Cook, Heteroscedastic G-optimal designs, *Journal of Royal Statistical Society, Series B*, 55, 871–880, 1993.

Wong, W. K., Recent advances in constrained optimal design strategies, *Statistical Neerlandica*, (Invited paper) 53, 257–276, 1999.

Woods, D. C., Robust designs for binary data: Applications of simulated annealing, *Journal of Statistical Computation and Simulation*, 80, 29–41, 2010.

Wynn, H. P., Results in the theory and construction of *D*-optimum experimental designs, *Journal of Royal Statistical Society, Series B*, 34, 133–147, 1972.

Yang, X. S., *Nature-Inspired Metaheuristic Algorithms*, Luniver Press: Frome, U.K., 2008.

Yang, X. S., *Nature-Inspired Metaheuristic Algorithms*, 2nd ed., Luniver Press, U.K., 2010.

Yang, M., S. Biedermann, and E. Tang, On optimal designs for nonlinear models: A general and efficient algorithm, *Journal of the American Statistical Association*, 108, 1411–1420, 2013.

Yang, J., A. Mandal, and D. Majumdar, Optimal designs for 2^k factorial experiments with binary response, *Statistica Sinica*, 2015. doi:10.5705/ss.2013.265.

Yu, Y., Monotonic convergence of a general algorithm for computing optimal designs, *Annals of Statistics*, 38, 1593–1606, 2010.

Yu, Y., *D*-optimal designs via a cocktail algorithm, *Statistics and Computing*, 21, 475–481, 2011.

Zhou, J., *D*-optimal minimax regression designs on discrete design space, *Journal of Statistical Planning and Inference*, 138, 4081–4092, 2008.