

**Advanced Experimental Design**  
On “The Coordinate-Exchange Algorithm for  
Constructing Exact Optimal Experimental  
Designs”

Chris Long

December 12, 2002

# Introduction

One of the fundamental problems in the design of experiments is the generation of efficient screening designs. The purpose of a screening design is to give a rough overall idea as to the shape of the experimental response surface, while only requiring a relatively small number of runs. This will then suggest areas for further exploration in order to find those experimental parameters that will result in an output optimal, or one close to optimal. To start we need a mathematical way to quantify the relative efficiency of screening designs, and there are several ways of doing this. This choice of measure will determine which types of screening designs are favored as well as the algorithms for choosing efficient screening designs.

## Good screening designs

In order to discuss good screening designs, we need a notion of exactly what "good" means in this context. Given a design matrix  $X \in \mathcal{X}$ , where  $\mathcal{X}$  is the feasible matrix design region, we're looking for a function  $F(X)$  that satisfies some desirable properties. As we're trying to minimize the total uncertainty of the response over the entire region it's natural to consider those functions  $F$  which depend only on the variance/covariance matrix  $D = (X'X)^{-1}$ .

Some reasonable basic properties for such an  $F$  are

$$F(D) \geq 0 \text{ for positive semidefinite } D \quad (1)$$

$$F(0) = 0 \quad (2)$$

$$F(cD) = cF(D) \text{ for } c \geq 0. \quad (3)$$

As we're trying to capture total uncertainty in the response, properties (1) and (2) are natural. Property (1) says that our measure of uncertainty should be nonnegative, as the variance/covariance matrix is always positive semidefinite. Furthermore, if the response is known with absolute certainty (i.e.  $D = 0$ ), then (2) says that  $F(D) = 0$ . If there is uncertainty in our response function, our measure of efficiency should reflect this; if there is no uncertainty in our response function, our measure should reflect this as well. Finally, property (3) is a normalization requirement in the sense that if all of the variances and covariances change by the same amount, then this should be reflected by  $F(D)$  changing by this amount as well. Note that (2) follows immediately from (3).

What are some examples of such functions  $F(D)$ ? The two most common examples lead to the notion of  $D$ -efficient and  $L$ -efficient (most commonly, trace-efficient).

For the first example, consider the function  $F(D) = |D|^{\frac{1}{p}}$ , where  $|D|$  is the determinant of  $D$ , and  $D$  is a  $p \times p$  matrix (and so there are  $p$  points in the design). Property (2) is immediate, and property (3) follows from the definition of the determinant of a matrix as the sum of signed products of length  $p$ . Finally, property (1) follows from the determinant of a matrix equalling the product of its eigenvalues, and a matrix being positive semidefinite if and only if all of its eigenvalues are real and nonnegative.

As this  $F(D)$  is defined via the determinant, this measure of efficiency will be referred to as  **$D$ -efficiency**.

For the second example mentioned, consider the function  $F(D) = \text{tr}(D)$ , where  $\text{tr}(D)$  is the trace of  $D$ . Here properties (2) and (3) are immediate, and property (1) follows from the trace of a matrix equalling the sum of its eigenvalues, and again a matrix being positive semidefinite if and only if all of its eigenvalues are real and nonnegative.

As this  $F(D)$  is defined via the trace, this measure of efficiency will be referred to as **trace-efficiency**. Note that any nonnegatively weighted sum of the matrix diagonal, with total weight 1, will satisfy the three basic properties as well. All of these fall in the more general category of  $L$ -efficiency, where here  $L$  refers to  $F(D)$  being a linear function of  $D$ .

Other  $F(D)$  that will satisfy all three basic properties would be let  $F_i(D)$  be the  $i$ th root of the sum of all the products of length  $i$  of the eigenvalues of  $D$ ; here  $F_p(D) = |D|^{\frac{1}{p}}$  and  $F_1(D) = \text{tr}(D)$ . Values for  $i$  other than  $p$  and 1 seem to be rarely (if ever) used, however. Appropriate  $F(D)$  may also be defined with respect to various matrix norms.

Good, efficient, and optimal designs all refer to designs with small values of  $F(D)$ , and all of these terms are used interchangeably in various papers that deal with this topic. The term **exact** refers to the best possible design in the feasible matrix design space. Mathematically,  $X$  is an **exact**  $F$ -optimal design if

$$F(D) = \min_{X \in \mathcal{X}} F(X),$$

where  $\mathcal{X}$  is the feasible matrix design space. There can be more than one such exact  $F$ -optimal design, but there will be at least one, as we're considering closed compact feasible design spaces and continuous  $F$ .

How should we compare two candidate screening designs under some  $F(D)$ ? This brings up the notion of **relative efficiency**. Relative efficiency of two designs under some  $F(D)$  is defined to be

$$r_e = \frac{F(D_1)}{F(D_0)},$$

where  $D_1$  is the variance/covariance matrix of our candidate design, and we're comparing it to a matrix with variance/covariance matrix  $D_0$ .

## Approximation algorithms

Unfortunately, in general finding exact optimal designs is hard. Finding exact optimal designs in general requires solving a large nonlinear mixed integer programming problem, as the number of feasible designs explodes rapidly as the number of factors and levels increases.

But we live in the real world, and we don't need the absolute best design, only one that's good enough. This is where approximation algorithms come in.

A typical approximation algorithm seeks to locate a good solution by the following sequential process:

1. Choose initial feasible solution (random/greedy)
2. Modify solution slightly (random/greedy)
3. Repeat 2. until finished, then report best solution seen

Random methods modify the current solution in some random way, and this change is accepted or rejected via some decision routine (worse solutions may be accepted under certain decision routines). Greedy methods modify the current solution in a way that improves the score; as they're seeking to improve the score for every iteration of the process they're frequently referred to as hillclimbing algorithms.

A classic, and very successful, approximation algorithm will illustrate these basic features, and will be useful as a contrast to the exchange algorithms considered later.

### Simulated annealing (Metropolis algorithm)

Simulated annealing is an example of a random approximation algorithm. An initial candidate solution is generated, either at random or via another approximation algorithm. The current candidate is modified in some small, random way, and then this (possibly less optimal) candidate is accepted with a probability based on the Boltzmann kernel  $\exp(-\Delta E/kT)$  [2]. The notion

of relative efficiency appears here via  $\Delta E$ . A new idea appears also, that of the temperature  $T$ . This is simply an increasing function of the number of current iterations; such a function is referred to as a cooling schedule and is a feature found in some random approximation algorithms.

Simulated annealing has been used very successfully in many areas of combinatorial optimization, most notably for the travelling salesman problem.

## Exchange algorithms for optimal designs

One large class of pure greedy algorithms for generating  $D$ -efficient designs are the exchange algorithms. Exchange algorithms hill climb by adding new design points and removing existing design points to improve the objective.

There are both Rank-1 and Rank-2 exchange algorithms, and these classifications are based on how the algorithm changes the points in the current candidate design matrix:

1. Rank-1: Choose points to add and delete sequentially (Wynn, DET-MAX)
2. Rank-2: Choose points to add and delete simultaneously (Fedorov, modified Fedorov,  $k$ -exchange,  $kl$ -exchange)

The algorithm we're interested in is a Rank-2 algorithm. In particular, it's based on the  $k$ -exchange algorithm. The  $k$ -exchange algorithm modifies the current candidate design via the process:

1. Examine  $k$  least critical points only
2. Least critical:  $x_i$  with smallest  $v(x_i)$ , where  $v(x) = f'(x) \cdot D \cdot f(x)$
3. Among these  $k$ , find the best exchange to make

Here  $x$  is a point in our  $q$ -dimensional design space, where the total number of factors is  $q$ , and  $f'(x)$  is the corresponding row of our design matrix  $X$ .

If  $k = 1$ , this is Wynn's algorithm; if  $k = n$ , this is the modified Fedorov algorithm. Generally,  $k \leq n/4$  appears to be a good choice.

In the  $D$ -efficiency setting, the  $k$ -exchange algorithm is computationally efficient, since if the  $i$ th point in the current candidate design is exchanged for another point  $x$ , the multiplicative change to  $|X'X|$  is given by

$$\Delta_D(x_i, x) = 1 + (v(x) - v(x_i)) + (v^2(x, x_i) - v(x)v(x_i)).$$

Note that the number of optimizations required is reduced by the fact that we're only considering the  $k$  least critical points in the design. We're only considering single-point exchanges here; multiple-point exchanges have been empirically shown to not be as effective as single-point exchanges. This agrees with a standard philosophy in approximation algorithms that many small steps are generally better than fewer but larger steps.

This philosophy hints that we might want to modify this algorithm even further.

## The coordinate-exchange algorithm

If smaller steps are generally better, why not try sub-single-point exchanges? This is precisely the idea behind the coordinate-exchange algorithm. It follows the procedure:

1. Again, choose  $k$  least critical points
2. Examine each point for the best coordinate to exchange
3. Make this best coordinate exchange

This algorithm is in fact shown by Meyer and Nachtsheim [1] to be faster than the standard  $k$ -exchange algorithm, while still leading to designs that are equally  $D$ -efficient. Whereas computing  $\Delta_D(x_i, x)$  for the standard  $k$ -exchange algorithm involves evaluating a  $p \times p$  quadratic form (recall  $D$  is a  $p \times p$  matrix), its evaluation will be a much smaller quadratic form for the coordinate-exchange algorithm. For example, for first order models only four multiplications and a single addition are necessary. For large  $p$  the savings are significant, and this is primarily what leads to the 1 or 2 orders of magnitude gain in speed that Meyer and Nachtheim observe.

# Bibliography

- [1] Meyer, Nachtsheim (1995), “The Coordinate-Exchange Algorithm for Constructing Exact Optimal Experimental Designs,” *Technometrics*, 37, 60-69.
- [2] Aarts, Korst Simulated Annealing and Boltzmann Machines, Wiley-Interscience (1989).