Genetic Algorithms Can Improve the Construction of D-Optimal Experimental Designs

J. POLAND, A. MITTERER*, K. KNÖDLER and A. ZELL
WSI Rechnerarchitektur
Universität Tübingen
Sand 1, D - 72076 Tübingen
GERMANY
poland@informatik.uni-tuebingen.de http://www-ra.informatik.uni-tuebingen.de

Abstract: - We study the benefits of Genetic Algorithms, in particular the crossover operator, in constructing experimental designs that are D-optimal. To this purpose, we use standard Monte Carlo algorithms such as DETMAX and k-exchange as the mutation operator in a Genetic Algorithm. Compared to the heuristics, our algorithms are slower but yield better results.

Key-Words: - Genetic Algorithm, Memetic Algorithm, Design of Experiments, DOE, D-Optimal, DETMAX Algorithm, k-Exchange Algorithm, Combinatorial Optimization.

1 Introduction

Design of experiments (DOE) has been of both great theoretical and practical interest. There exists a number of criteria for optimality of designs, and there are several algorithms for constructing optimal designs. The most common and most widely used criterion is D-Optimality.

Given are n candidates $x_1, \dots, x_n$, defined by points $u_1, \dots, u_n$ in the input space (the d-dimensional euclidean space) and a regression type (e.g. a second order polynomial). For a choice of $p < n$ candidates $\xi = (j_1, \dots, j_p) \in \{1 \dots n\}^p$, we write $|\xi| = p$ and define the design matrix

$$X_\xi = (x_{j_1} \dots x_{j_p})',$$

i.e. X_ξ is the matrix formed of the chosen candidates. We now consider the model

$$y = X_\xi \beta + \epsilon$$

which is linear in the coefficients, where ϵ is a random vector with distribution $N(0, \sigma^2 \cdot Id)$ and y is the observation vector of size p. Then,

$$\hat{\beta} = (X'_\xi X_\xi)^{-1} X'_\xi y$$

is the least squares estimate for β and has covariance matrix $(X'_\xi X_\xi)^{-1}\sigma^2$. To obtain a model of high quality, one should choose the candidates $j_1, \dots, j_p$ in order to minimize this covariance matrix. Several notions of minimality have been considered. D-Optimality means to minimize the determinant $\det((X'_\xi X_\xi)^{-1})$ or, equivalently, maximize $\det(X'_\xi X_\xi)$. The desired number of points in the design is usually a fixed number p_0.

We remark that this approach applies to arbitrary polynomial models or, even more general, to models that are linear in the coefficients. For example, a second order polynomial model for the points $(u_{1,j}, u_{2,j})_{j=1}^p$ is obtained by $x_j = (1\ u_{1,j}\ u_{2,j}\ u_{1,j}^2\ u_{2,j}^2\ u_{1,j} \cdot u_{2,j})'$. Low order polynomial models have many practical applications in engineering, e.g. they are used to imitate the behaviour of complex systems. In this case, the process of measurement which leads to the observation vector y may be quite expensive, and one is interested in the best possible design, while the expenses to find the design (computer time) are almost negligible.

2 Common algorithms

Almost all algorithms for constructing D-optimal designs are Monte Carlo algorithms, heuristics, that base on the idea of sequentially exchanging "bad" candidates of a design for "better" ones. A comparison of these algorithms can be found in [1]. In this section, we outline two of these algorithms, which we will use as reference and as mutation operators for the genetic algorithms.

*A. Mitterer, BMW Group, D - 80788 München, Germany, alexander.mitterer@bmw.de

2.1 DETMAX

The DETMAX algorithm was first suggested in 1974 by Mitchell ([7]) and subsequently improved (see e.g. [2]). One starts with a random design $\xi = (j_1, \ldots, j_p)$. One also initializes the *failure set* F to the empty set. F is supposed to contain all those designs which did not lead to an improvement. (In fact, for space-saving reasons, F will contain the determinant of those designs.)

In each step, with p denoting the actual number of candidates, there are three cases:

- $p = p_0$. Then one randomly decides whether to add or to remove a candidate.
- $p > p_0$. In this case, a candidate is removed, if the current design is not in F. Otherwise, a candidate is added.
- $p < p_0$. In this case, a candidate is added, if the current design is not in F. Otherwise, a candidate is removed.

After that, one has to decide, which candidate is added or removed. Since

$$\det\left((X'_\xi \; x) \begin{pmatrix} X_\xi \\ x' \end{pmatrix} \right) = \det(X'_\xi X_\xi) \cdot (1 + x'(X'_\xi X_\xi)^{-1} x),$$

the candidate x_j is added for which $1 + x'_j(X'_\xi X_\xi)^{-1}x_j$ attains its maximum. If a candidate is removed, this term is replaced by $1 - x'_j(X'_\xi X_\xi)^{-1}x_j$.

There are some more updating formulas which allow to perform also the other computations (such as updating $(X'_\xi X_\xi)^{-1}$) with a small effort (see [2] for details). Because of the accumulation of floating point errors, one should perform a bootstrapping after a number of steps, i.e. calculate the exact values, we used 10 steps.

When after one step the actual number of candidates p equals the desired number p_0, one checks if there is an improvement of the determinant. In this case, the failure set F is emptied, otherwise all designs that were met up to now are added to F.

The algorithm terminates if $|p - p_0| > q$, where q is a constant. Mitchell proposed $q = 6$, which we adopted. A former version of this algorithm using $q = 1$ often gets stuck in local minima with poor determinant. When $|p - p_0| > 1$, it is called an "excursion".

2.2 k-exchange

Another heuristic for constructing D-optimal designs is called k-exchange algorithm (see e.g. [5]). It is based on the idea that it might not be optimal to add the candidate for which $1 + x'_j(X'_\xi X_\xi)^{-1}x_j$ attains its maximum. If afterwards a candidate is removed, these two steps can be considered as one step (the *exchange* of a candidate), thus yielding more improvement. Of course, this increases the size of the problem considerably. Instead of finding the maximum of p terms, now roughly $N \cdot p$ have to be examined. To keep this number lower, one can consider only the best k candidates for addition and the worst k candidates for removal. Thus, there are k^2 terms of which the maximum is to be found.

3 The crossover operator

Especially in large scale cases with a great number of candidates, the above exchange procedures are likely to get stuck in a local optimum which is a result of a suboptimal design in a part of the input space. Another design may be better for this part, but worse for another. If the better part of the first design is *combined* with the better part of the second, there could be hope that the resulting design "inherits" the good properties of both designs. Hence, we consider a crossover operator that takes two designs and builds two new designs of them. For this aim, we first have to define an appropriate representation of a design.

Consider a *bit string* b of length n, $b = b_1 b_2 \ldots b_n$. The candidate j occurs in the design b if and only if $b_j = 1$. Thus, it is possible to apply an arbitrary standard type of crossover (one point, two point, uniform) to two designs b_1 and b_2, obtaining two new designs b_3 and b_4 (see e.g [3] or [4]).

This binary representation has a severe drawback. In particular if $p \ll n$, each design b uses much space to encode little information, similar to a sparse matrix. Therefore, another representation can be used. A design corresponds to a set c coded as an ordered *list* containing all points of the design. For a genetic algorithm all the lists should have a fixed length, but on the other hand during the process of optimization it is desirable to try and combine designs of different size. Therefore, the lists have the fixed length of $2 \cdot p_0$, and the unused entries are filled with 0. Note that the *alphabet* used for this representation is no longer binary, but has size $n + 1$.

Of course, this list representation requires a different crossover operator. It has turned out that the simulation of a standard crossover operator on the binary representation works well. In addition, this simulation can easily be done in time $O(p)$, while the standard crossover operator on the binary representation needs time $O(n)$.

More explicitely, the uniform crossover operator on two lists c_1 and c_2 producing c_3 and c_4 reads as follows. Take the smaller of the first elements of c_1 or c_2 and remove it. With probability $\frac{1}{2}$, add it to c_3 or c_4, respectively. If the first elements of c_1 and c_2 are equal, remove them from both c_1 and c_2 and add them to both c_3 and c_4. Repeat these steps until both c_1 and c_2 are empty.

4 Repetitions

The list representation of designs has one more advantage. If p_0 is not too small, the optimal design may contain repetitions, i.e. candidates that are contained two or more times. This fact seems not to be very intuitive at a first glance, but it can be illustrated by a simple example. Consider a linear model in one dimension with $n = 10$ equidistant candidate points. Then the best way to choose $p_0 = 4$ candidates is to take the minimum point and the maximum point each twice, as one can easily verify.

With the list representation, we are able to encode designs that contain repetitions, while this is not possible with the binary representation. Even if k is small enough such that the optimal design does not contain repetitions, it has shown out that admitting repetitions during the search accelerates the algorithm.

5 Genetic Algorithms

We have already described the representation of the individuals, i.e. either binary or list coding, as well as the crossover operator for the genetic algorithm. The fitness function is almost obvious, one roughly takes the inverse of the determinant of $X'_\xi X_\xi$. Of course a larger number of actual design points p than the desired number p_0 allows a larger determinant. Therefore we perform an initial *estimate* d_0 of the optimal inverse determinant by once applying the heuristic. Then we define the fitness function

$$\begin{aligned} \phi(\xi) &= \det(X'_\xi X_\xi)^{-1} \\ &+ C(t) \cdot 1_{|\xi|>p_0} \cdot (|\xi| - p_0) \cdot d_0, \end{aligned}$$

where $C(t)$ is a function dependent of the actual generation t of the genetic algorithm. We used an increasing sigmoid function having $C(0) = 1$ and $C(t_{max}) = 20$, where t_{max} is the number of generations that are performed. Thus, the fitness function is non-stationary (see e.g. [6]), but in each generation t its minimum will have $|\xi| = p_0$, since

$$\min_{|\xi|>p_0} \phi(\xi) \geq d_0 \geq \min_{|\xi|=p_0} \phi(\xi).$$

Moreover, the function is non-decreasing in t. This non-stationary fitness function has the effect that in the early phase of the genetic algorithm designs with more points are tolerated, allowing a larger genetic variety, while later those designs are eleminated, increasing the chance for an optimal design with the desired $|\xi| = p_0$.

For the mutation, we pursue a similar strategy. While in the late generations it is important to draw the maximum effect out of the heuristic, we can save time in the early phase by aborting the heuristic after a certain number of steps and choosing a small q (in case of the DETMAX algorithm) or k (in case of the k-exchange algorithm).

Finally, for generating the initial population, we use the heuristic for about one third of the population. The number of design points p is choosen randomly for each individual between p_0 and $p_0 + 10$, where $p = p_0$ has the greatest probability. Again we can save time by aborting the heuristic after a certain number of steps and choosing a small q (for DETMAX) or k (for k-exchange). The rest of the population is generated randomly with p design points, where p is uniformly distributed in $\{p_0, \dots, p_0 + 10\}$.

6 Experimental results

The algorithms have been tested with several candidate sets, varying number of design points and polynomial models of different order. Here, the results of a test data set with 9^4 candidates are presented, defined by a full factorial set in 4 dimensions with 9 points for each coordinate plus a small random perturbation. Different test data sets produced very similar, partly almost identical, output. We show two "tight" designs, a 3rd order model having 35 design points and a 4th order model having 70 design points, which is in fact the lowest possible number of points for the respective model. Moreover, we present two "large" designs, 3rd and 4th order with 100 and 150 points, respectively. In addition, we present two "real world" data sets from an industrial application (see [8]).

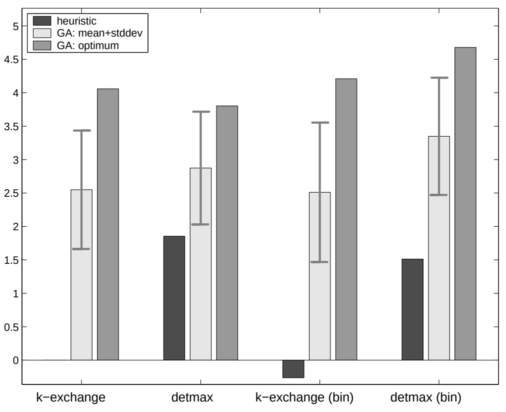

Fig. 1. Test data set, tight design, 3rd order

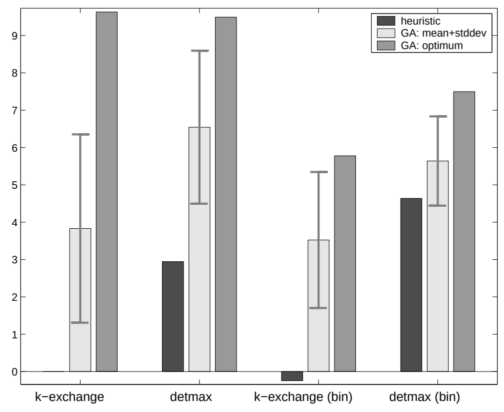

Fig. 3. Test data set, tight design, 4th order

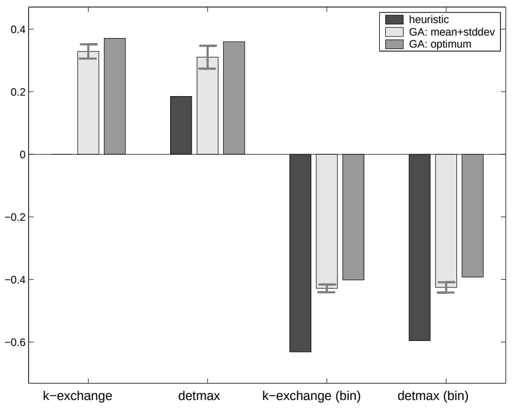

Fig. 2. Test data set, large design, 3rd order

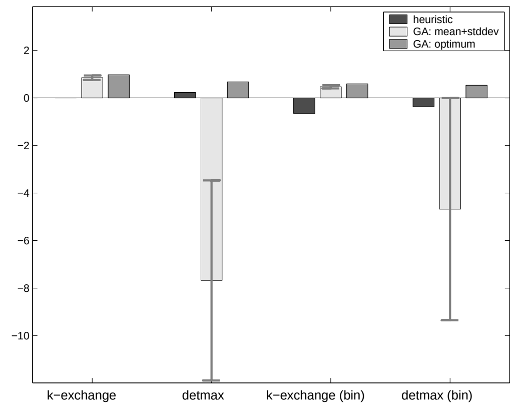

Fig. 4. Test data set, large design, 4th order

Each figure contains the relative performances of the four heuristics, k-exchange, DETMAX, binary k-exchange and binary DETMAX, and in comparison, the performance of the genetic algorithm using these heuristics as mutation operators. It is common to execute the heuristic a number of times R and choose the best resulting design, we used $R = 20$. Also the genetic algorithms have been evaluated 20 times, the figures display the average performances with standard deviations and the best performances. The plots use a logarithmic (base 2) scale, the values are relative to the fitness of the pure k-exchange, a positive value means a better (lower) determinant. Thus, the leftmost bar (k-exchange heuristic) is the reference and is always 0, and a bar of height -1 means that the fitness value of the corresponding measurement was twice the reference value.

We observe that the genetic algorithm yields always better designs than the corresponding heuristic. Especially for large designs, the list version performs considerably better than the binary versions. In one case (Fig. 4), the DETMAX mutation operator seems not to produce a robust genetic algorithm. The best design is mostly found by the genetic algorithm with k-exchange.

In our MATLAB implementation, one run of a heuristic takes, depending on the data set, about 2 seconds using k-exchange and about 4 seconds using DETMAX. One call of the genetic algorithm with a population of $\mu = 40$ individuals running for $t_{max} = 100$ generations takes on the average about 170 seconds with k-exchange and about 240 seconds with DETMAX. Thus, the running time of the genetic algorithms is roughly 80 times the running time of the corresponding heuristic.

7 Conclusions

Genetic algorithms can improve the construction of D-optimal experimental designs. Particularly when measuring is expensive, the presented genetic algorithms can save costs. Moreover, they can provide a way to obtain good *practical* designs, that are good with respect to *more than one* optimality criterion. This approach could lead towards more

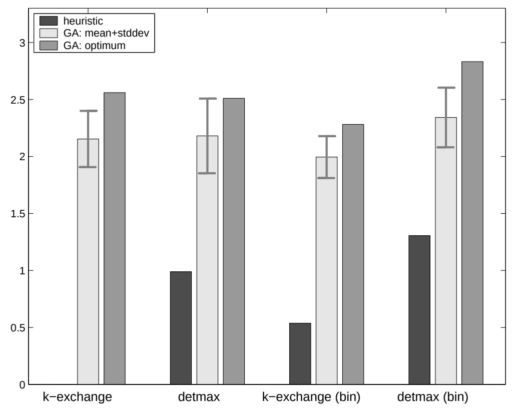

Fig. 5. Real data set 1, rather tight design, 3rd order

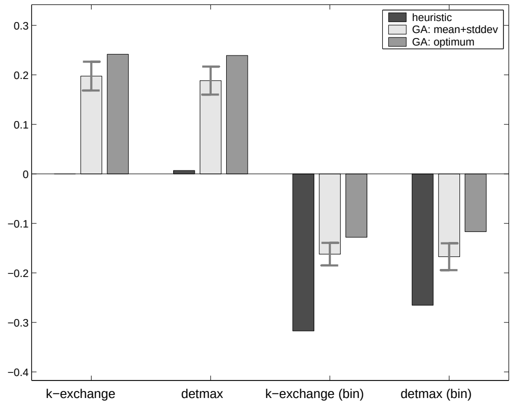

Fig. 6. Real data set 2, large design, 3rd order

powerful design algorithms, meeting the demands of the growing complexity of todays real world systems.

Acknowledgments

We thank Thomas Fleischhauer and Frank Zuber-Goos for helpful discussions. This research has been supported by the BMBF (grant no. 01 IB 805 A/1).

References:

[1] R. D. Cook and C. J. Nachtsheim. A comparison of algorithms for constructing exact d-optimal designs. *Technometrics*, 22(3):315–323, Aug 1980.

[2] Z. Galil and J. Kiefer. Time- and space-saving computer methods, related to mitchell's detmax, for finding d-optimum designs. *Technometrics*, 22(3):301–313, Aug 1980.

[3] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[4] J. Holland. *Adaptions in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press, 1975.

[5] M. E. Johnson and C. J. Nachtsheim. Some guidelines for constructing exact d-opitmal designs on convex design spaces. *Technometrics*, 25:271–277, 1983.

[6] Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. In R. G. Reynolds J. R. McDonnell and D. B. Fogel, editors, *Fourth Annual Conference on Evolutionary Programming*, Cambridge, MA, 1995.

[7] T. J. Mitchell. An algorithm for the construction of "d-optimal" experimental designs. *Technometrics*, 16(2):203–210, May 1974.

[8] A. Mitterer. *Optimierung vielparametriger Systeme in der Antriebsentwicklung, Statistische Versuchsplanung und Künstliche Neuronale Netze in der Steuergeräteauslegung zur Motorabstimmung.* PhD thesis, Lehrstuhl für Meßsystem- und Sensortechnik, TU München, 2000.