Review Article

Quality and Reliability Engineering International

(wileyonlinelibrary.com) DOI: 10.1002/qre.1591

Published online 17 January 2014 in Wiley Online Library

Using Genetic Algorithms to Design Experiments: A Review

C. Devon Lin,[a*†] Christine M. Anderson-Cook,[b] Michael S. Hamada,[b] Leslie M. Moore[b] and Randy R. Sitter[c]

Genetic algorithms (GAs) have been used in many disciplines to optimize solutions for a broad range of problems. In the last 20 years, the statistical literature has seen an increase in the use and study of this optimization algorithm for generating optimal designs in a diverse set of experimental settings. These efforts are due in part to an interest in implementing a novel methodology as well as the hope that careful application of elements of the GA framework to the unique aspects of a designed experiment problem might lead to an efficient means of finding improved or optimal designs. In this paper, we explore the merits of using this approach, some of the aspects of design that make it a unique application relative to other optimization scenarios, and discuss elements which should be considered for an effective implementation. We conclude that the current GA implementations can, but do not always, provide a competitive methodology to produce substantial gains over standard optimal design strategies. We consider both the probability of finding a globally optimal design as well as the computational efficiency of this approach. Copyright © 2014 John Wiley & Sons, Ltd.

Keywords: design criteria; design representation; optimization algorithm; orthogonal array; response surface design; screening design; search algorithm

1. Introduction

While many standard designs exist for a broad class of experimental situations, there are scenarios required when creating a design to satisfy particular objectives or to conform to logistical constraints. It is common to use computer search algorithms to create an optimal design for cases when there is a specialized objective, a nonstandard-sized experiment, to constraint on the design space, or to augment an existing design. With today's computing power, the experimenter is free to tailor the criterion over which to optimize to best reflect the goals of the experiment.

We begin by defining a design as a matrix whose rows correspond to runs or blocks and columns to factors, where the matrix elements or cells are factor levels for a given run of the experiment (row) for a particular factor or input variable (column). There are a few variations on this structure for characterizing a design, including Angelis,[1] which treated blocks as columns.

Deciding on the parameters of the design of experiment problem includes considering how many and which factors should be considered, what is the design space for the problem, what size of the experiment is sought, and whether there are constraints on the factor levels or factor level combinations that can be run during the experiment. In addition, the experimenter should consider what objective or criterion is best, given the overall goal of the experiment. Once these issues have been considered, then the decision about which algorithm to use to find the best design should be considered, and this is often a difficult decision. Wolpert and Macready[2] demonstrate the 'no free lunch' theorem, which says that no optimization algorithm can dominate when performance is averaged across all possible scenarios. In other words, there is no universal winner for optimization algorithms, and the choice of which algorithm is best must be a function of the particular characteristic of the problem to be solved. Hence, there is a need for a thoughtful implementation that matches the key features of the algorithm with the defining aspects of the problem to be solved. One of the more popular algorithms being used in recent years for creating experimental designs has been the genetic algorithm. The goals of this paper are to provide a review of the original genetic algorithm and its variants that have been used for finding optimal experimental designs in the recent literature, to address several important issues associated with this optimization problem, and to discuss some challenges and opportunities in this research area. We explore the appropriateness and highlight some of the strengths and weaknesses of using genetic algorithms (GAs) for this class of optimization problems, that is, finding optimal experimental

[a]*Queen's University, Kingston, Ontario, Canada*
[b]*Los Alamos National Laboratory, Los Alamos, New Mexico, USA*
[c]*Simon Fraser University, Burnaby, British Columbia, Canada*
**Correspondence to: C. Devon Lin, Queen's University, Kingston, Ontario, Canada*
†*E-mail: cdlin@mast.queensu.ca*

designs. We provide examples in which GAs successfully provide satisfactory solutions and examples in which GAs fail to efficiently find optimal or nearly optimal designs. We draw conclusions about how we should be assessing the success of the method. Here, it is not our intent to provide a practical guide for researchers who want to apply GAs to any optimization problem in the experimental design. Rather, we highlight unresolved issues in their implementation and understanding of what the GAs' components achieve.

We begin with a brief overview of GAs, which are motivated by evolutionary biology. (refer to Goldberg[3] and Michalewicz[4]) A population of individuals is characterized by their *chromosomes*. A chromosome consists of *genes*, which across individuals is expressed in different degrees—the early GA literature simplified the degree of gene expression to 'off' and 'on'. The goodness of an individual is measured for the optimization criterion by its *fitness*. *Crossover* and *mutation* change the population of individuals over time. Crossover occurs when individuals *mate*, where the chromosomes of children are generated stochastically from their parents' chromosomes. Based on the idea of survival of the fittest, a good implementation of the algorithm fosters parents with higher fitness being more likely to mate and produce better children. Mutation occurs when a few genes are randomly changed, for example, from on to off or vice versa. A GA begins by randomly generating a population—the original population or 0th generation. Crossover and mutation operators are applied to generate additional individuals. There are too many variants of crossover and mutation to list here, but some are discussed later in this paper. Ideally, these steps to generate new population members should be strategic and should be formulated to have reasonable probability of producing children who are superior to their parents according to the criterion of interest. Through *selection*, evolution occurs where the population for the next generation is chosen from individuals from the previous generation plus the additional individuals generated by crossover and mutation. There are several selection variants. For example, one scheme is *elitist* by taking the best individuals according to their fitness, whereas another selects individuals with probability proportional to their fitness. There are also variants on mutation rates. One variant, called *punctuated equilibrium* and motivated by evolutionary biology, mimics mutations tending to occur less often over time, but every once in a while, there is an upset where the mutations start to occur more frequently than before. Typically, the probability of a mutation occurring decreases from generation to generation, whereas punctuated equilibrium resets the probability to the starting probability at a fixed frequency. Note that some of the literature uses 'selection' to mean the choice of parents to crossover and uses 'replacement' to mean selection as we have defined it. Also, the original GA used genes whose bits or binary variables were motivated by the off and on expression of genes. Subsequent GAs used genes with real-valued variables when the problem being solved called for their use.

In applying GAs to designing experiments, the user has to decide on what *representation* or *encoding* to use. This involves specifying what a chromosome represents (typically a design) and what a gene represents. Common choices for a gene are a run (row), a factor (column), or a factor level for a given run (cell of design matrix). Then, the user specifies the crossover, mutation, and selection operations to complete the GA description. Tables I and II show that many papers since the early 1990s have used GAs to create and select different 'optimal' experiments. Table I includes papers that represent standard applications of designing experiments, such as finding D-optimal designs. Table II presents more specialized applications such as designing microarray experiments, although there are some connections with those in Table I. GAs have also been used to optimize response surface predictions from the designed experiments that Alvarez *et al*.[37] reviews. Alvarez *et al*.[37] also lists some of the applications of GAs to designing experiments that we do but does not provide a comprehensive review of issues. In the remainder of this paper, we refer to papers by the bracketed numbers in Tables I and II. For example, Safadi and Wang (1991)[5] will be referred to as [1].

To better understand alternatives to GAs as well as hybrids of GAs, we briefly discuss several other standard choices: exchange algorithm, simulated annealing (SA) algorithm, and tabu search. Point and coordinate exchange (CE) algorithms typically begin with a random design and then iteratively consider improvements to the design by replacing either a single run (i.e., a point) or a single factor level within a run (i.e., a design matrix cell referred to as a coordinate) to improve the design by making the locally optimal choice at each stage. Both point and CE algorithms are general and can be applied to problems with different types of factors without difficulty, but point exchange algorithms rely on a finite candidate set of possibilities to consider at each step. Incorporating continuous factor levels is difficult and typically involves discretizing to a grid, that is, a finite number of possibilities. Both algorithms can be applied without adaptations regardless of the design criteria. For certain criteria, CE can be made very fast through the use of low-rank update formulas and other tricks that leverage the inherent sparsity of the problem. This means that many random starts can be considered, which often mitigates the potential problem that a given start can lead to a local optimal or suboptimal design. See Fedorov[38] and Cook and Nachtsheim[39] for more details.

Another possible alternative is SA. By analogy with the physical process of annealing in metallurgy, each step of the algorithm considers alternate designs that are randomly generated as being 'nearby' to the current solution. Alternatives are chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called the temperature), that is, gradually decreased during the process. See Haines[40] for more details. Papers [2] and [10] use a combination of GA and SA to find optimal designs. Modifications to the algorithm have been proposed to avoid becoming trapped at local optima. Other papers that have used SA for design optimization include Meyer and Nachtsheim,[41] Morris and Mitchell,[42] Fang *et al*.,[43]and Wit, Nobile, and Khanin.[44] In the nonstatistical literature, there is a considerable discussion about how to tune the SA parameters to achieve a good balance between early search of the entire space and later comprehensive search of the local region near promising candidate solutions.

Another option that has been used in the statistics literature is the tabu search. This search is commonly used for solving combinatorial optimization problems and was adapted by Jung and Yum[45] for finding optimal designs. This approach uses a local neighborhood search, with special machinery to prevent returning to designs that have already been explored.

The focus of this paper is the chromosomal representation of a statistical experimental design and the genetic operations, crossover and mutation, used for finding optimal designs. Noting the recent publication of numerous papers using GAs for designing experiments, we consider what elements of the implementation should be summarized for the reader, how well the chromosomal

Table I. Standard applications

Paper	Problem	Criterion	Approach/gene	Notes
[1] Safadi & Wang (1991)[5]	Mixed-level OA	Number of unbalanced level pairs, see Wang and Safadi (1990)[6]	Column permutation of elements	
[2] Govaerts & Sanchez-Rubal (1992)[7]	RSM	D	run	Crossover exchange, mutation SA exchange, candidate list
[3] Broudiscou *et al*. (1996)[8]	Mixed-level	D main effects	Matrix cell	Smaller run size than OA
[4] Montepiedra *et al*. (1998)[9]	RSM	D	Matrix cell, bit representation	Discretized ER
[5] Hamada *et al*. (2001)[10]	RSM, nonlinear and logistic regression	Bayesian EIG	Run	Run crossover, element mutation
[6] Poland *et al*. (2001)[11]	RSM	D	Run binary, list run representations	Candidate list, run crossover, and undetermined mutation
[7] Angelis (2003)[1]	Incomplete block design	A	Block	
[8] Borkowski (2003)[12]	RSM	A, D, G, I	Run	Run LC crossover, element mutation variants
[9] Heredia-Langner *et al*. (2003)[13]	RSM mixture/ process	D	Matrix element	Constrained ER
[10] Drain *et al*. (2004)[14]	RSM, RPD	Scaled PEV and SEV, desirability index	run	Constrained ER and GA-SA hybrid
[11] Heredia-Langner*et al*. (2004)[15]	Robust RSM mixture	Scaled D for model class, desirability index	Matrix element	Constrained ER
[12] Goldfarb *et al*. (2005)[16]	Robust mixture, process, noise, mixture	G	Matrix element, same as [9]	Constrained ER
[13] Park *et al*. (2005)[17]	RSM	G	Matrix element, same as [9]	Cuboidal ER
[14] Park *et al*. (2006)[18]	RSM	G	Binary, list run representations	Cuboidal ER, cost constrained
[15] Guo *et al*. (2007)[19] also Guo (2003)[20]	Mixed-level design	Balance coefficient, J_2, criteria LC	Binary, list run representations	Smaller run size than OA, candidate list
[16] Rodriguez *et al*. (2009)[21]	Robust RSM, mixture	Scaled PEV and SEV, desirability index	Matrix element, same as [9]	Various ER, cost constrained

EIG: expected information gain; ER: experimental region; LC: linear combination; OA: orthogonal array; PEV: prediction error variance; RPD: robust parameter design; RSM: response surface model; SA: simulated annealing; SEV: slope estimation variance.

representations used captured the essence of a design, and to understand what the crossover and mutation operations accomplish in generating new designs. Our review is framed in terms of issues which we briefly list next and then present in separate sections. We conclude the paper with a discussion. Some of the key issues in applying GAs for designing experiments are as follows:

1. Good chromosomal representation of a design
2. Formulation of advantageous crossover and mutation operations
3. GA performance and assessment

2. Chromosomal representation

The challenge in applying a GA to a particular problem is to develop a relevant chromosomal representation. Such a challenge is not shared by other search algorithms. By defining the chromosomes and genes to match strategically defined crossover and mutations steps, we can hope to increase the chance of finding good designs within the class of designs sought and to increase the computational efficiency of the algorithm to find the optimal design. If a poor implementation is selected, the proposed designs in

Table II. Specialized applications

Paper	Problem	Notes
[17] Cela *et al*. (2000)[22]	Supersaturated experiment	$E(S^2)$, *n0* and *m0* criteria, small even run size designs, select columns from balanced 2-level columns
[18] Bashir & Simpson (2002)[23]	Supersaturated experiment	$E(S^2)$ criterion, select subset of columns from half-fraction of Hadamard matrix
[19] Bates *et al*. (2003)[24]	Computer experiment	Repulsive force criterion, *n* run Latin hypercube sampling design, column as permutation of *n* factor levels, two representations
[20] Chen *et al*. (2003)[25]	Follow-up design	model discrimination criterion, matrix element, follow-up two-level fractional factorial
[21] Marseguerra *et al*. (2003)[26]	Degradation tests	Percentile precision for total cost within budget, standard multivariate optimization problem, see also Hamada *et al*. (2008)[27]
[22] Wagner & Nichols (2003)[28] [WN2003], Kao *et al*. (2009)[29] [K2009]	fMRI experiment	A criterion [WN2003], multi-objective optimization [K2009], block design, stimulus levels as treatments, see [7]
[23] Liefvendahl & Stocki (2006)[30]	Computer experiment	Distance/repulsive force criteria, *n* run Latin hypercube sampling design, column as permutation of *n* factor levels
[24] Sexton *et al*. (2006)[31], Anthony & Keane (2004)[32]	Assembled products	D criterion, column as a permutation of available parts, two representations and corresponding genetic algorithm operations
[25] Tveit & Fogelholm (2006)[33]	Multistage experiment	D criterion, first order model, factor level in a given stage constrained by factor level in preceding stage
[26] Koukouvinos *et al*. (2007)[34]	Supersaturated experiment	$E(S^2)$ criterion, *k*-circulant design, select *k*-circulant generator
[27] Gondro & Kinghorn (2008)[35]	Microarrays	Weighted multi-objective criteria, incomplete block design with blocks of size 2, additional two-level factor within blocks, more complex than [7]
[28] Crombecq & Dhaene (2010)[36]	Sequential computer experiments	Add one run at a time, space filling, lacks genetic algorithm details

new generations may not stay within the desired class of designs, which will lead to the need to repair or discard new candidates, or may substantially diminish the efficiency of the search. In addition, a poor implementation might not likely yield promising candidates, also reducing search efficiency.

Most papers in Table I for standard applications (except [1]) consider a population of designs so that the chromosome represents an individual design and the genes represent runs (or blocks) or factor levels. So far, whatever a gene has been chosen to represent, the genetic analogy begins to break down because in genetics the gene is unique and has a particular function or ordering. However, for most designs, there is the notion of isomorphism—we have the same design whether we interchange rows; and depending on the problem, we may have the same design whether columns or factor level labels are interchanged. That is, the currently defined role of a gene (i.e., row or design matrix cell) is only manifested in the presence of other genes. As we discuss in the next section, this breakdown impacts the intent and effectiveness of the crossover operation.

Regarding gene encoding, an early application of GAs to designing experiments, [4], uses multiple binary variables or bits to represent a factor level, for example, a seven-level factor requires three bits. However, a gene as a bit implies that some factor levels are more naturally ordered as close to each other. For example, 000 (level 1) and 100 (level 5) are a single bit flip apart, while 000 (level 1) and 011 (level 4) are two bit flips apart. Also, because of the particular number of factor levels, 111 may arise from crossover and mutation operations and has to be repaired to 000–110 (levels 1–7). Consider a crossover step that combines genes of two instances 101 and 011. If we split after the first bit, taking the first bit from the first instance and the second and third bits from the second instance, we obtain the infeasible result of 111. Also, the real-valued level has to be translated to and from bits, increasing the computational burden. Manipulating the real-valued levels directly avoids the need for translation and repair.

The chromosomal representation should complement the criterion for which the design is being optimized. For example, orthogonal arrays (OAs) or nearly orthogonal arrays (NOAs) have properties that are connected to the columns of the design matrix, and hence, we might expect that a GA that manipulates the columns of the design would be more advantageous than focusing on the rows. The approach of [1] seeks mixed-level OAs of n runs for m factors by sequentially adding one column at a time via a GA. A GA population is a collection of chromosomes, each of which is a binary vector of length $n(n-1)/2$; and a gene indicates whether the ith and jth row entries of the current design column should be switched or not. For $k = 1,\ldots,m$, the initial choice for the kth column is a balanced column (a necessary condition for OAs). The current kth column is then updated by applying the best chromosome in the jth generation of the GA to the kth column previously obtained using the best chromosome in the $(j-1)$th generation. In contrast with [1]'s column-based approach, [15] uses a run-based (row) approach, where the crossover step can result in repeated runs, and balanced columns are not guaranteed. Papers [19], [23], and [24] are other examples that make attempts at a suitable representation

for the type of design being sought. Papers [19] and [23] construct optimal Latin hypercube sampling designs, and [24] constructs optimal designs for assembled problems, whose designs are matrices with columns that are permutations of a set of integers.

Of course, choosing a criterion best suited to the goal is desirable and critically important for solving the right problem. Paper [3] uses the D-criterion for a main-effects model rather than those suited to OAs used by [1] and [15]. In [3]'s defense, an OA is D-optimal, and [4]'s intent was to find small designs for main-effects only models with run sizes much smaller than required by an OA. The criterion should be chosen to reflect the top priority of the experiment and to allow for quantitative assessment of the goodness of the design. For an approach to consider more than one criterion, see Lu, Anderson-Cook, and Robinson.[46]

If there is knowledge or theory that a particular structure or class of designs is optimal, then a GA should search within that class. For example, [1] considers designs with the run size equal to the minimum run size required by an OA and works with balanced columns—both are necessary conditions for an OA. For D-optimal, G-optimal, and I-optimal design criteria, we would expect that changing locations in the design space would have an impact on the goodness of a design so that a run-based chromosomal representation makes sense (the majority of Table I applications except [1], [3], and [15] are run based).

Several papers consider discrete design spaces. Papers [2] and [6] choose designs from a candidate set of runs, and both represent a design by a list of runs from the candidate set, for example, if there are N runs with identifiers 1,…, N, then a design of n runs consists of a list of n identifiers. Paper [6] also considers a binary representation of N bits, with the ith run being included in the design if the ith bit is 1. For a rectangular experimental space (i.e., without constraints in the experimental space), [4] represents a discretized cuboidal experimental space without a list; using m bits, a factor's range can be discretized into 2^m values by evaluating the m bits and dividing by 2^m—to obtain values between 0 and 1. Choice of runs for mixed-level OAs or NOAs can be viewed as a candidate set, albeit a large one. However, [15] considers small mixed-level designs so the candidate set is small and uses both a list representation such as [2], [6], and [14], as well as a binary representation such as [6] and [14].

Approaches for discrete design spaces (i.e., factors with a fixed number of levels) are likely to have a connection to predefined candidate sets, which may either define possible runs of the experiment (point-exchange-based algorithms) or possible levels of each factor (coordinate-exchange-based algorithms). The implementation of these types of designs is likely to be quite different from designs where the design space and possible factor values are continuous. This discretizing of the factor levels simplifies the search for an optimal design, but may exclude the true overall optimum from consideration. In the continuous paradigm, we have the option to convert our space to a discrete space. One significant potential advantage of the candidate set approach occurs when the design space is non-cuboidal with constraints on possible design locations. In this case, once an appropriate candidate set has been identified, there is no longer a need to verify that the suggested children in subsequent generations of the GA are within the allowable design space, as both the crossover and mutation steps are defined to select from only allowable runs in the candidate set.

The specialized applications in Table II (not specifically discussed thus far) appear to use appropriate representations for the designs they consider, which focus on less general experimental constraints or scenarios. See the individual papers for more details.

3. Formulating advantageous crossover and mutation operations

What does the standard crossover operation do? By standard crossover operation, we mean that the chromosomes of two parents are randomly partitioned, and the parents trade their first pieces to generate two new individuals. We claim that because the isomorphism of designs is not captured by any current chromosomal representation, the standard crossover is essentially a random global jump. Paper [6], which generates designs from a candidate set of runs, suggests first ordering the runs within a design by their identifiers and performing a crossover by randomly choosing a run from the first ordered run of each of the parents, a run from the second ordered run of each of the parents, and so on; for a large set of candidates, the ordering would seem to do little to address the isomorphism of designs. So the crossover operation appears to amount to a random search, much like generating the initial generation that was entirely random. This differs from optimization in nondesign settings where each gene plays a unique role, and the criterion of interest is typically more continuous. In these cases, taking a combination or interpolated value based on two good solutions makes intuitive sense and has a good chance that it will result in improvement, or at least a good value of the criterion.

Is there a version of crossover that makes more intuitive sense for increasing the likelihood of producing an ideal child based on the characteristics of the parents? Paper [2] used an exchange algorithm treating the better parent as the current design and the other parent as the set of candidates. While this exchange algorithm requires more time (perhaps substantially more for large designs), this crossover operation ensures that the child is no worse than the best parent. One paper has commented on [2] as not being a 'real GA', but its crossover operation better achieves the intent of a crossover operation—better children from good parents—than other more standard definitions of this step.

We next comment on the connection between the crossover operation and the gene representation. In defining the representation to be used, the gene and crossover operations should be complementary and should be allowed for the strong possibility of generating viable promising children. In [9]–[12], which use a matrix representation for a design in a constrained space specified by a set of candidates, the gene needs to be a run (row) in order for the crossover operation to automatically maintain the constraint. If cells in the design matrix are combined with the crossover (i.e., the gene as a cell), then it is possible to create infeasible locations outside the design space. Another alternative is a coordinate crossover, where a random cell in the design matrix is selected, and cell values above and to the left are switched between parents. This alternative does not automatically satisfy any constraints nor address isomorphism.

In the next section, we consider a crossover based on the base-s representation of each row of two parents, where s is the number of levels of each factor. For example, the base-2 representation of a row (1,0,1,0) in a two-level design of four factors is 10. One

crossover with this representation works as follows. Suppose D_1 and D_2 are two parents. Let $d_1 = (d_{11},\ldots,d_{1n})$ and $d_2 = (d_{21},\ldots,d_{2n})$, where d_{1i} and d_{2i} are the base-s representation of the ith row of D_1 and D_2, respectively. We then combine the vectors d_1 and d_2 and obtain the vector d by ordering the combined vector. Take the rows of odd indexes as one offspring and the rows of even indexes as the other offspring. This crossover tends to separate close points and thus spreads out the points in the resulting designs, a desirable operator for obtaining OAs; for examples, it tends to prevent replicates. We call a GA with such a crossover the row-order-based GA. The performance of the row-order-based GA for a specific design scenario is compared with others in the next section.

The mutation operation can be viewed as a mechanism for conducting a local search. In [9]–[12], mutation is achieved by randomly replacing a run in a design with another from the set of candidates to maintain any constraints. So, *localness* is defined by changing one run in the design. Paper [2] also randomly chooses from a candidate set but employs SA so that a run that leads to a poorer design can be accepted probabilistically to avoid being too greedy at any particular step of the algorithm. Other papers (e.g., [5]) perturb individual matrix cells that can be viewed as even more of a local search than replacing an entire run. When there is a set of candidates, if neighborhood information were to be constructed, mutation by choosing one of its neighbors would be more local than that achieved in [2], [6], and [9]–[12]. Paper [8] proposed a number of mutation variants in the form of random jumps, for example, switching signs, moving to an extreme from a cell value that is near the center, and vice versa. For mutation, paper [6] uses an abbreviated k-exchange or DETMAX algorithm to improve a candidate design. Finally, among the papers listed in Tables I and II, some papers apply mutation to the children generated by crossover, whereas others apply mutation to the parents to generate new children. Some papers do not precisely define how the mutation step is implemented.

Variants of GAs have considered a constrained design space, which are also handled by exchange algorithms, SA, and tabu search. Point-exchange algorithms utilize a candidate set that accommodates the constraints, and thus, no extra care must be taken in the procedure. For other search algorithms, including GAs, designs generated should be checked to meet the constraints. For GAs, special attention is needed when generating initial designs as well as new designs through crossover and mutation to avoid generating runs outside the constrained space. Examples of constrained spaces include $x_1 + x_2 + x_3 \leq 10$ for a three-factor experiment or $x_1 + x_2 + x_3 = 1$ for a three-factor mixture experiment (where x_1, x_2, and x_3 are proportions). Initial random designs can be generated by discarding runs outside the constrained space until the required number of runs is reached. A row-based crossover will also generate proper designs. Mutation may produce a run outside the constrained space, but it could be repaired, or mutation could continue until a proper run is generated. In the case of a mixture, proportions could be rescaled so that their total equals 1. Another possibility is to mutate all the proportions using a Dirichlet random variable centered at the current proportions. More complicated fixes are needed for individual constraints on the factors. For the $x_1 + x_2 + x_3 \leq 10$ constrained space, rescaling might work depending on whether there are individual constraints on the factors. As an alternative, [9] heavily penalizes the fitness of a design with runs outside the constrained space.

4. GA performance and assessment

For publication in peer-reviewed journals, it is typically important to demonstrate that the implemented GA performs well for the example or the more general class of problems considered. One fundamental criterion for determining the goodness of an algorithm is the ability to find the best solution, which in many situations may be unknown. Sometimes, there is a known bound on the design criterion that a design might attain. However, such a bound is not a definite standard, as the bound might not be attainable. There are also situations when no known bound exists so that judging the performance of an algorithm is even more difficult. The best design found by the algorithm for a particular criterion can be compared with existing designs for the same design scenario if they exist. But when a new criterion is being evaluated or a novel experimental design problem is being considered, how does one assesses the goodness of the best solution found by the algorithm? It may be asking too much to compare it with those designs obtained from other algorithms. However, a quick check is to compare it with a large number of random designs that are easy to generate.

In practice, one would hope to find a design that is close enough to the optimal design for the user's purposes in a timely manner. Because exhaustive searches are impractical for even moderately sized design problems, finding a 'best' design often means that while we wish to find the true optimal design, finding a design that is close to the true optimal design is sufficient for our experimental purposes. Often, there are many designs that perform quite close to the optimal design with little or no practical difference in performance from the true optimum.

Hence, in reporting the performance of a particular implementation of a GA for a particular problem with a specified criterion, we would recommend the following elements be required for publication in a peer-reviewed source as an evaluation framework:

1. Details about the implementation of the algorithm, including definition of the chromosomes and genes, number of generations, population size within a generation, initial generation, mechanism for crossover and mutation, selection mechanism for each generation, handling of constraints (if any).
2. Stopping rule for when the algorithm is deemed to have completed optimization (e.g., successive large improvements have ended).
3. Comparison of existing designs or those generated by other competitors for optimization, including possible alternative implementations of the GA, in terms of performance efficiency (value of design criterion achieved) and time efficiency (time or number of designs evaluated), including whether a known bound for the optimum criterion exists. At a minimum, the GA results should be compared with those from an equivalent number of random designs.

Regarding point 1, there is considerable variability in the amount of detail provided by the papers listed in Tables I and II. From a very few of the descriptions, it would be possible to reasonably reconstruct the algorithm, but for the majority of the papers, there are

key missing details that would prevent the reader from being able to reproduce the results. Both for easing the burden on a reader who wishes to use the algorithm in their own application and for validation of the methods, providing such details is important. Regarding point 2, at a minimum, the number of generations that the GA was ran needs to be stated. Also a plot of the criterion for the best design by generation, a so-called evolution plot, is useful in assessing convergence, which some of the papers (e.g., [7], [22], and [25]) provide.

Regarding point 3, Table III summarizes the performance of the GAs in Table I. Most of the papers listed in Tables I and II address performance, but only a few address time efficiency ([4], [11], and [24]). For example in some cases, there are dramatic savings in time efficiency while achieving near optimal performance; whereas in other cases, an exchange algorithm is more time efficient. Many papers show that GAs can nearly achieve the known optimal design or best existing design (e.g., [1], [2], [4], [5], [7], [9], [10], and [26]). Other papers show that GAs provide modest improvement over existing designs (e.g., [7], [8], [9], [11]–[14], [20], some cases of [23], and [26]). Apparently, [6] claims that GAs substantially outperform two types of exchange algorithms but does not provide enough details to verify or reproduce the results. Paper [23] shows that a competitor outperforms GAs in some cases it considered. Finally, other papers (e.g., [3], [21], [25], [27], and [28]) did not provide any performance comparison.

To illustrate the GAs success in finding a near-optimal design, we revisit Example 3 of [5] in light of the proposed evaluation framework given earlier. Their example considers a three-factor quadratic response model (10 regression coefficients β including an intercept) and a normal error with unknown variance σ^2. The run-based (i.e., run as a gene) GA maximizes Bayesian expected information gain (EIG) whose integral is approximated by Monte Carlo. We used the same GA as [5], whose details are given in their appendix. The (β, σ^2) prior and GA parameters are specified in [5]'s Example 3 (e.g., population size $M = 10$), except that we used $L = 1{,}000{,}000$ Monte Carlo simulations to evaluate EIG. We ran the GA for 900 generations employing punctuated equilibrium in batches of 100, that is, the crossover and mutation parameters were reset at the start of each batch. The EIG evolution plot for the 900 generations is displayed in Figure 1. We stopped after observing no change after two batches of 100 generations and found the first design displayed in Figure 2, whose EIG is 26.07; in fact, this design was found at generation 679. We see from Figure 1 that few small improvements occurred after generation 400. By looking at the design points, we noticed that their values are near -1.67, 0, and 1.67, where ± 1.67 for each factor defines boundary of the experimental region. This design appears as the second design in Figure 2, whose EIG is 26.13, and whose symmetry is particularly pleasing. As a quick comparison with the GA which evaluated $10 + 900 \times (10 + 10) = 18{,}010$ designs, we generated 18,010 random designs. The best of the random designs is shown as the third design in Figure 2, and its EIG is 22.18. Moreover, it does not suggest a better design that the GA design did.

Now, we present a small case study on using GAs to finding OAs and NOAs. We selected this problem to demonstrate performance of different implementations of GAs and to provide a cautionary tale about the importance of thinking carefully about the implementation and algorithm details based on a careful match of the design criterion and the encoding structure. Here, we consider strength two OAs, where for each pair of columns, every factor level combination appears the same number of times. For an NOA, every factor level combination appears nearly the same number of times.

In finding OAs and NOAs, we adopt J_2-optimality introduced in Xu.[47] The J_2-optimality requires the rows of a design to be as dissimilar as possible. The dissimilarity of any two rows is quantified by the Hamming distance, the number of positions at which the corresponding entries are different. Consider an $n \times m$ matrix $D = (d_{ij})$. For $1 \le i,j \le n$ let $\delta_{i,j}(D) = \sum_{k=1}^{m} w_k \delta(d_{ik}, d_{jk})$, where w_k is the weight of kth column, $\delta(u,v) = 1$ if $u = v$ and 0 otherwise. The J_2 criterion is given by $J_2(D) = \sum_{1 \le i < j \le n} [\delta_{ij}(D)]^2$. A design is J_2-optimal if it minimizes J_2. The J_2 of design D whose kth column has S_k levels, and weight w_k has a lower bound $L(m) = 2^{-1}\left[\left(\sum_{k=1}^{m} n w_k / s_k\right)^2 + \left(\sum_{k=1}^{m} (s_k - 1)(n w_k / s_k)^2\right) - n\left(\sum_{k=1}^{m} w_k\right)^2\right]$. Note that OAs can be constructed by combinatorial methods to attain the J_2 lower bound $L(m)$. NOAs do not attain the J_2 lower bound $L(m)$, but a value closer to the bound means that the design is more nearly orthogonal.

Initially in the study, we considered three GAs, a row-based GA, a row-order-based GA, and a column-based GA, in which M is the population size, that is, the number of parents, and the J_2 with the natural weights $w_k = S_k$ is used.

1. The row-based GA follows [5]. M initial designs are randomly generated, and their J_2 are evaluated. Two parents are chosen from the current M designs with the probability proportional to the reciprocal of their respective J_2, and crossover of the chosen parents is carried out by randomly selecting a run from parent pairs (i.e., first runs of both parents, second runs of both parents, etc.). M pairs of parents are chosen that generate M children. Mutation is performed on the M parents, thereby generating M additional designs: each matrix element has a chance of being mutated with probability $e^{-\mu g}$ (for generation g and mutation rate μ), and if mutated, a random choice is made from the factor levels excluding the current factor level. The best M designs out of the M parent designs and $2M$ generated designs make up the M parent designs for the next generation.
2. The row-order-based GA is the same as the row-based GA except that the crossover first orders the runs of the two parents by treating a run (row) as a base-s representation and then selects odd runs from ordered combined parents to form an offspring and even runs to form the other offspring.
3. For the column-based GA, initial random designs are generated using random balanced columns (or random nearly balanced columns if the desired run size does not permit balanced columns). Parents are chosen with the probability proportional to reciprocal of their respective J_2, and crossover is accomplished by randomly selecting columns from parent pairs (i.e., first column of both parents, second column of both parents, etc.). M pairs of parents are chosen to generate M children. Mutation is performed on the M parents generating an additional M designs, where each column is mutated with probability $e^{-\mu g}$; and if it is mutated, a random row is chosen, and the column entries including and above the random row selected and the remaining column entries are switched.

Table III. Relative performance efficiency comparison of three GAs with some competitors for obtaining orthogonal arrays

			Row-based genetic algorithm[a]		Row-ordered-based genetic algorithm[b]		Column-based genetic algorithm[c]		Safadi-Wang[d]				
n	*m*	*s*	I[e]	II[f]	I	II	I	II	I	II	Random[g]	Xu[h]	Lower bound
12	6	2	1.00(0.23)	0.985(0)	1.00(0.64)	1.00(0.12)	1.00(0.97)	1.00(0.99)	1.00(1.00)	1.00(1.00)	1.00	1.00	2160
20	7	2	1.00(0.001)	0.986(0)	1.00(0.037)	0.998(0)	1.00(0.498)	1.00(0.587)	1.00(0.420)	1.00(0.431)	0.997	1.00	9240
12	11	2	0.988(0)	0.958(0)	0.988(0)	0.969(0)	1.00(0.287)	1.00(0.328)	1.00(0.999)	1.00(0.995)	0.965	1.00	6600
20	19	2	0.980(0)	0.971(0)	0.980(0)	0.991(0)	0.991(0)	0.988(0)	0.700(0)	0.700(0)	0.968	1.00	61560
27	13	3	0.930(0)	0.856(0)	0.929(0)	0.927(0)	0.958(0)	0.951(0)	0.472(0)	0.525(0)	0.898	1.00	50544
100	20	5	0.895(0)	0.847(0)	0.885(0)	0.885(0)	0.932(0)	0.917(0)	0.235(0)	0.235(0)	0.876	0.971	1900000

The proportion of 1000 trials in which an orthogonal array was found is given in parentheses.
[a]Row-based genetic algorithm (GA) is the GA in [5].
[b]Row-order-based GA is the GA using the mutation in [5] and the crossover using base-*s* representation.
[c]Column-based GA is the GA using the mutation in [5] and the crossover applied on columns of parents.
[d]Safadi-Wang algorithm [1] using J_2 criterion.
[e]I refers to the case of mutation without punctuated equilibrium.
[f]II refers to the case of mutation with punctuated equilibrium.
[g]Random balanced columns.
[h]Xu (2002)[47] algorithm using J_2 criterion.

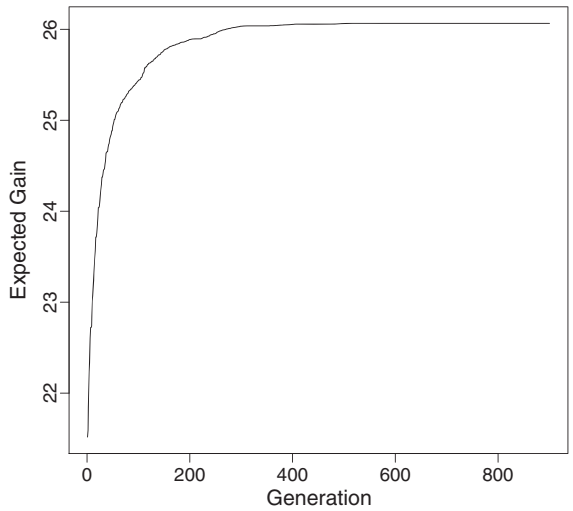

Figure 1. Expected information gain trace for [5]'s Example 3 over 900 generations

We chose these three GAs because the row-based ones ignore the column-based nature of OAs and NOAs, which the column-based one takes into account. The row-order-based GA attempts to avoid replicates so that it might perform better than the row-based one.

First, we consider how well these three GAs find OAs. Table III compares the three types of GAs with 500 generations ($G = 500$), a population size of 100 per generation ($M = 100$), and periodical mutation probabilities of $e^{-\mu g}$ in generation g with $\mu = 0.01$. In each type of GA, mutations without and with punctuated equilibrium are considered. The punctuated equilibrium is accomplished by executing each GA in successive batches of 100 generations. Table III displays relative performance efficiency (RPE) of the best design found for each of the approaches. We define the RPE as $J_{2,best}^{-1}/J_{2,opt}^{-1}$, where $J_{2,opt}$ is the lower bound in the last column of Table III, and

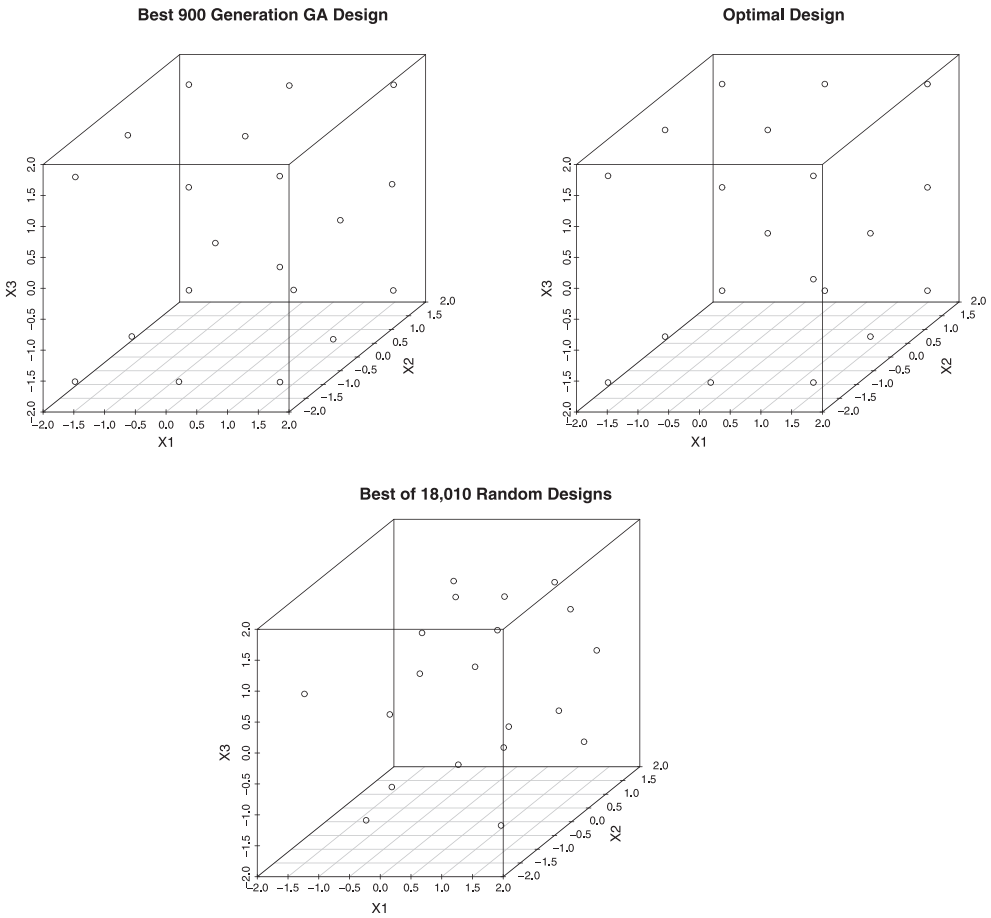

Figure 2. Best 900 generation genetic algorithm design, optimal design, and best of 18,010 random designs whose expected information gain are 26.07, 26.13, and 22.18, respectively

$J_{2,best}$ is the criterion value for the best design found by the GA among 1000 trials of the GA. That is, an RPE of 1 means that the lower bound has been achieved. In parentheses, we show the proportion of times that the GA finds an *s*-level OA with *n* runs and m factors.

Table III indicates that searching for small OAs such as 12 runs with 6 factors, the GAs have a good chance of finding it. As the number of runs and factors increase, OAs are not found even with 1000 starts of the algorithm, and the RPEs of the best obtained designs decrease; the RPEs for the best GA version are generally about 0.90 or higher, however. For smaller design problems, the column-based GA outperforms the row-based ones in terms of the frequency in finding an OA. However, for OAs of moderate and large run sizes, the column-based GA is only slightly better.

In addition, punctuated equilibrium (version II in Table III) does not necessarily improve the performance of GA when the number of generations is small. It can be useful when the number of generations is large and the mutation rate decreases rapidly as shown for one trial in Figure 3 for a five-level designs of 100 runs with 20 factors, where $\mu = 0.04$, and each GA version is executed in successive batches of 400 generations. Figure 3 indicates that both row-based GA and column-based GA with punctuated equilibrium yield smaller J_2 than the counterparts without punctuated equilibrium. However, in Figure 4, we see when $\mu = 0.01$ that only the column-based GA with punctuated equilibrium yields smaller J_2 than its counterpart without punctuated equilibrium; also for this case, the row-based and row-ordered-based GAs do much better without punctuated equilibrium.

Table IV presents the results for the same three GAs for finding some selected NOAs. In these cases where the lower bound may not be achievable, the RPEs of the best version of the GAs are moderately high although somewhat less than those in Table III for the OAs; the RPEs for the five-level NOAs are substantially worse than those for the smaller level designs.

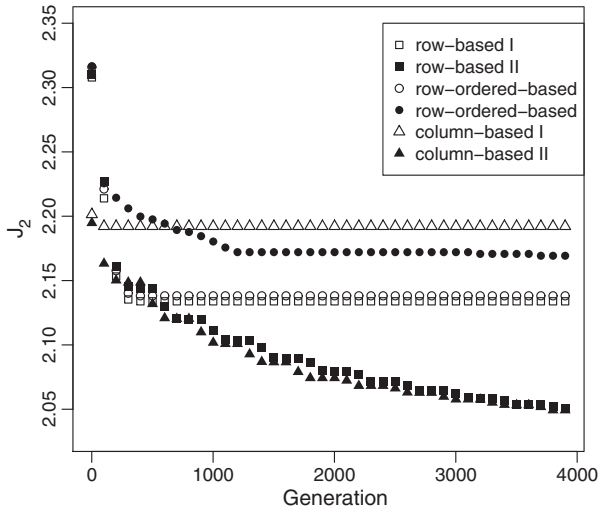

Figure 3. J_2 of five-level designs of 100 runs with 20 factors obtained by row-based genetic algorithm (GA) without punctuated equilibrium (I), row-based GA with punctuated equilibrium (II), row-order-based GA without punctuated equilibrium (I), row-order-based GA with punctuated equilibrium (II) column-based GA without punctuated equilibrium (I), and column-based GA with punctuated equilibrium (II) when $\mu = 0.04$

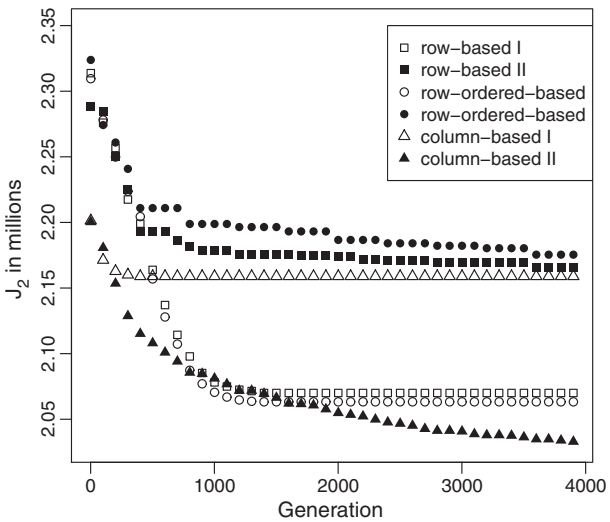

Figure 4. J_2 of five-level designs of 100 runs with 20 factors obtained by row-based GA without punctuated equilibrium (I), row-based GA with punctuated equilibrium (II), row-order-based GA without punctuated equilibrium (I), row-order-based GA with punctuated equilibrium (II) column-based GA without punctuated equilibrium (I), and column-based GA with punctuated equilibrium (II) when $\mu = 0.01$

In the previous section, we mentioned that a comparison against random designs provides a quick check of the performance of the GAs. We see from the third last column of Tables III and IV that the three GAs provide only a modest improvement over randomly generated designs with balanced columns. Note that the number of random designs generated is the same as the number of designs evaluated by the GAs to provide a fair comparison. Because we view the crossover as basically generating random designs, it appears that the modest improvement of the GAs is coming from mutation, that is, local optimization.

The three GAs have varying degrees of suitability for the problem considered, which naturally suggests a column-based approach. The two row-based GAs certainly are not a natural choice, and the column-based GA amounts to locally optimization of random designs.

An alternative for constructing OAs and NOAs is a column-based algorithm that successively adds columns. Xu[47] proposed such a method that does a selective systematic search because considering all possible columns for large designs is intractable. We see from the second to the last column of Tables III and IV that Xu's algorithm based on 1000 trials is consistently superior especially for the second NOA in Table IV.

In a different use of GAs, [1] proposed a stochastic version of successively adding columns for mixed-level OAs of n runs for m factors. In their approach, a GA population is a collection of chromosomes each of which is a binary vector of length $n(n-1)/2$ whose genes correspond to the row pairs (1,2), …, (1,n), (2,3),… , (2,n),…, ($n-1$,n); a gene value of 1 indicates that the row entries of the current column are switched. We see that the chromosomes represent permutations of the initial balanced column. Consequently, a GA is run for each added column. The first column of the design is any balanced column. For $k = 2, \ldots, m$, the initial column for the kth column of the design is any random balanced column (a necessary condition for OAs). The current kth column is then updated by applying the best chromosome in the jth generation of the GA to the kth column previously obtained using the best chromosome in the $(j-1)$th generation. Here, we use the same population size $M = 100$, the number of generations $G = 500$, and $\mu = 0.01$ as we did with the three other GAs. See columns 10 and 11 of Tables III and IV for the disappointing results we obtained using [1]'s GA with the J_2 criterion based on 1000 trials. We see in Table V for one trial for the 20-run OA with 19 two-level factors that once the lower bound is not achieved (here as the eighth column is added), the performance of the GA degrades quickly as more columns are added.

Paper [15] used a row-based approach for constructing small mixed-level arrays related to our row-order-based approach that chooses design points from a full factorial design. (However, their run ordering and crossover are different.) They also used a hybrid criterion that weights how balanced a design and a standardized J_2 are. (They omitted details about their GA and hybrid criterion, which make it difficult to reproduce an implementation of the GA that they used.) They compared the GA with a row-coordinate-exchange algorithm[41] and Xu's algorithm.[47] For the 15-run design with four factors (2, 3, 5, 7 levels), all three algorithms obtained the same design. For the remaining 19 designs, the row-coordinate-exchange and Xu's algorithm found better designs than the GA in terms of J_2, with Xu's algorithm performing slightly better.

5. Discussion

There has been much work in the literature demonstrating the use of GAs for designing experiments of various types. We have reviewed this work and discussed various implementation issues including the choice of gene in the chromosomal representation of a design and the potentially advantageous definition of the crossover and mutation operators. For example, careful thought is required in specifying the chromosomal representation of a design and the definition of a gene, for example, a column as a gene seems natural for J_2 for nearly OAs; whereas for other criteria such as D optimality, a row as a gene is more natural. A good choice for a gene also helps the crossover operation and can avoid having to repair generated inadmissible solutions.

We recommend that adequate details of a published GA be provided so that it can be validated and replicated. Also, its performance needs to be comprehensively addressed and at a minimum compared to randomly generated designs. Some of the cases in the literature suggest that GAs can provide modest but not substantial improvements, while in our case study, GAs performed somewhat better than random designs but not as well as a strategic systematic competitor. GAs are well suited to continuous design spaces and provide an alternative to exchange algorithms when the number of candidate runs becomes

Table IV. Relative performance efficiency comparison of three genetic arrays with some competitors for obtaining nearly orthogonal arrays

n	m	s	Row-based genetic algorithm[a]		Row-ordered-based algorithm[b]		Column-based algorithm[c]		Safadi-Wang[d]		Random[g]	Xu[h]	Lower bound
			I[e]	II[f]	I	II	I	II	I	II			
27	15	3	0.925	0.876	0.921	0.935	0.943	0.936	0.436	0.436	0.889	0.998	65610
50	13	5	0.802	0.718	0.809	0.801	0.862	0.841	0.232	0.232	0.766	0.908	170625
64	20	4	0.901	0.854	0.898	0.900	0.942	0.927	0.303	0.303	0.888	0.972	737280

[a]Row-based genetic algorithm (GA) is the GA in [5].
[b]Row-order-based GA is the GA using the mutation in [5] and the crossover using base-s representation.
[c]Column-based GA is the GA using the mutation in [5] and the crossover using columns of parents.
[d]Safadi-Wang algorithm [1] using J_2 criterion.
[e]I refers to the case of mutation without punctuated equilibrium.
[f]II refers to the case of mutation with punctuated equilibrium.
[g]Random balanced columns.
[h]Xu (2002)[47] algorithm using J_2 criterion.

Table V. Best J_2 and the lower bound for one trial of the [1]'s genetic algorithm

Column	Best J_2	Lower Bound
2	1040 (1.000)	1040
3	2040 (1.000)	2040
4	3360 (1.000)	3360
5	5000 (1.000)	5000
6	6960 (1.000)	6960
7	9240 (1.000)	9240
8	11856 (0.999)	11840
9	15176 (0.973)	14760
10	19216 (0.937)	18000
11	23976 (0.899)	21560
12	29456 (0.864)	25440
13	35656 (0.831)	29640
14	42576 (0.802)	34160
15	50216 (0.777)	39000
16	58576 (0.754)	44160
17	67656 (0.734)	49640
18	77440 (0.716)	55440
19	87976 (0.700)	61560

intractable. However, GAs have their own implementation issues mentioned earlier as well as deciding on the population size, mutation parameter, and how many generations to run the GA.

What remains elusive is a chromosomal representation that achieves the intent of crossover (i.e., good parents produce better children) and accounts for the isomorphism of a design. We challenge researchers to pursue this direction. In addition, we also encourage research in quantifying the separate benefits of the crossover and mutation operators; for example, is crossover more than a random search for design optimization? Another problem is the effect of fitness functions on searching optimal designs. One might seek designs based on one specific criterion, while other relevant criteria might be more efficient. Finally, we see opportunities for developing GAs for different design structures (such as for split-plots) and for different models (such as generalized linear models).

Acknowledgements

We thank C.C. Essix for her encouragement and support of this paper. We note with deep regret that our dear colleague Randy Sitter was tragically lost at sea on September 19, 2007. Lin is supported by the Natural Sciences and Engineering Research Council of Canada.

References

1. Angelis L. An evolutionary algorithm for a-optimal incomplete block designs. *Journal of Statistical Computation and Simulation* 2003; **73**:753–771.
2. Wolpert DH, Macready WG. No free lunch theorems for optimization. *IEEE Transactions of Evolutionary Computation* 1997; **1**:67–82.
3. Goldberg DE. Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley: Reading, MA, 1989.
4. Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Springer-Verlag: New York, NY, 1992.
5. Safadi RB, Wang RH. The use of genetic algorithms in the construction of mixed multilevel orthogonal arrays. Computing Science and Statistics: Proceedings 23rd Symposium on the Interface, Keramidas EM, Kaufman SM (eds.) Interface Foundation of North America: New York, NY, 1991; 322–325.
6. Wang RH, Safadi RB. Generating mixed multilevel orthogonal arrays by simulated annealing. Computing Science and Statistics: Proceedings 22nd Symposium on the Interface, Page C, LePage R (eds.) Springer-Verlag: New York, 1990; 557–560.
7. Govaerts B, Sanchez-Rubal P. Construction of exact D-optimal designs for linear regression models using genetic algorithms. *Belgian Journal of Operations Research, Statistics and Computer Science* 1992; **1–2**:153–174.
8. Broudiscou A, Leardi R, Phan-Tan-Luu R. Genetic algorithm as a tool for selection of D-optimal design. *Chemometrics and Intelligent Laboratory Systems* 1996; **35**:105–116.
9. Montepiedra G, Myers D, Yeh AB. Application of genetic algorithms to the construction of exact D-optimal designs. *Journal of Applied Statistics* 1998; **22**:817–826.
10. Hamada M, Martz HF, Reese CS, Wilson AG. Finding near-optimal Bayesian experimental designs via genetic algorithms. *The American Statistician* 2001; **55**:175–181.
11. Poland J, Mitterer A, Knödler K, Zell A. Genetic algorithms can improve the construction of D-optimal experimental designs. *Advances In Fuzzy Systems and Evolutionary Computation* (Proceedings of WSES EC 2001), Mastorakis N (ed.) World Scientific Engineering Society Press: Danvers, 2001; 227–231.
12. Borkowski JJ. Using a genetic algorithm to generate small exact response surface designs. *Journal of Probability and Statistical Science* 2003; **1**:65–88.
13. Heredia-Langner A, Carlyle WM, Montgomery DC, Borror CM, Runger GC. Genetic algorithms for the construction of D-optimal designs. *Journal of Quality Technology* 2003; **35**:28–46.
14. Drain D, Carlyle WM, Montgomery DC, Borror CM, Anderson-Cook C. A genetic algorithm hybrid for constructing optimal response surface designs. *Quality and Reliability Engineering International* 2003; **20**:637–650.

15. Heredia-Langner A, Montgomery DC, Carlyle WM, Borror CM. Model-robust optimal designs: a genetic algorithm approach. *Journal of Quality Technology* 2004; **36**:263–279.
16. Goldfarb HB, Borror CM, Montgomery DC, Anderson-Cook CM. Using genetic algorithms to generate mixture-process experimental designs involving control and noise variables. *Journal of Quality Technology* 2005; **37**:60–74.
17. Park Y, Richardson DE, Montgomery DC, Ozol-Godfrey A, Borror CM, Anderson-Cook CM. Prediction variance properties of second-order designs for cuboidal regions. *Journal of Quality Technology* 2005; **37**:253–266.
18. Park Y, Montgomery DC, Fowler JW, Borror CM. Cost-constrained G-efficient response surface designs for cuboidal regions. *Quality and Reliability Engineering International* 2006; **22**:121–139.
19. Guo Y. Simpson JR, Pignatiello JJ, Jr. Construction of efficient mixed-level fractional factorial designs. *Journal of Quality Technology* 2007; **39**:241–257.
20. Guo Y. Construction of efficient fractional factorial designs. Florida State University Master's Thesis, 2003.
21. Rodriguez M, Montgomery DC, Borror CM. Generating experimental designs involving control and noise variables using genetic algorithms. *Quality and Reliability Engineering International* 2009; **25**:1045–1065.
22. Cela R, Martinez E, Carro AM. Supersaturated experimental designs. New approaches to building and using it: part I. Building optimal supersaturated designs by means of evolutionary algorithms. *Chemometrics and Intelligent Laboratory Systems* 2000; **52**:167–182.
23. Bashir AA, Simpson JR. Finding the important factors among many factors in designed experiments. unpublished manuscript, 2002.
24. Bates SJ, Sienze J, Langley DS. Formulation of the Audze–Eglais uniform Latin hypercube design of experiments. *Advances in Engineering Software* 2003; **34**:493–506.
25. Chen C-L, Lin R-H, Zhang J. Genetic algorithms for MD-optimal follow-up designs. *Computers & Operations Research* 2003; **30**:233–252.
26. Marseguerra M, Zio E, Cipollone M. Designing optima degradation tests via multi-objective genetic algorithms. *Reliability Engineering and System Safety* 2003; **79**:87–94.
27. Hamada MS, Wilson AG, Reese CS, Martz HF. Bayesian Reliability, Springer: New York, NY, 2008.
28. Wagner TD, Nichols TE. Optimization of experimental designs in fMRI: a general framework using a genetic algorithm. *NeuroImage* 2003; **18**:293–309.
29. Kao M-H, Mandal A, Lazar N, Stufken J. Multi-objective optimal experimental designs for event-related fMRI studies. *NeuroImage* 2009; **44**:849–856.
30. Liefvendahl M, Stocki R. A study on algorithms for optimization of Latin hypercubes. *Journal of Statistical Planning and Inference* 2006; **136**:3231–3247.
31. Sexton CJ, Anthony DK, Lewis SM, Please CP, Keane AJ. Design of experiment algorithms for assembled products. *Journal of Quality Technology* 2006; **38**:298–308.
32. Anthony DK, Keane AJ. Genetic algorithms for design of experiments on assembled products. University of Southampton School of Mathematics Technical Report No. 385, 2004.
33. Tveit T-M, Fogelholm C-J. Multi-period steam turbine network optimisation. Part I: simulation based regression models and an evolutionary algorithm for finding D-optimal designs. *Applied Thermal Engineering* 2006; **26**:993–1000.
34. Koukouvinos C, Mylona K, Simos DE. Exploring k-circulant supersaturated designs via genetic algorithms. *Computational Statistics & Data Analysis* 2007; **51**:2958–2968.
35. Gondro C, Kinghorn BP. Optimization of cDNA microarray experimental designs using an evolutionary algorithm. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 2008; **5**:630–638.
36. Crombecq K, Dhaene T. Generating sequential space-filling designs using genetic algorithms and Monte Carlo methods. Lecture Notes in Computer Science, **6457**. Springer-Verlag: Berlin, 2010; 80–88.
37. Alvarez MJ, Ilzarbe L, Viles E, Tanco M. The use of genetic algorithms in response surface methodology. *Quality Technology and Quantitative Management* 2009; **6**:295–307.
38. Fedorov VV. Theory of Optimal Experiments. Academic Press: New York, NY, 1972.
39. Cook RD, Nachtsheim CJ. A comparison of algorithms for constructing exact D-optimal designs. *Technometrics* 1980; **22**:315–324.
40. Haines LM. The application of the annealing algorithm to the construction of exact optimal designs for linear-regression models. *Technometrics* 1987; **29**:439–447.
41. Meyer RK, Nachtsheim CJ. Simulated annealing in the construction of exact optimal design of experiments. *American Journal of Mathematical and Management Science* 1988; **8**:329–359.
42. Morris MD, Mitchell TJ. Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference* 1995; **43**:381–402.
43. Fang KT, Ma CX, Winker P. Centered L_2-discrepancy of random sampling and Latin hypercube design and construction of uniform designs. *Mathematical Computation* 2002; **72**:275–296.
44. Wit E, Nobile A, Khanin R. Near-optimal designs for dual channel microarray studies. *Applied Statistics* 2005; **54**:817–830.
45. Jung JS, Yum BJ. Construction of exact D-optimal designs by tabu search. *Computational Statistics and Data Analysis* 1996; **21**:181–191.
46. Lu L, Anderson-Cook CM, Robinson TJ. Optimization of designed experiments based on multiple criteria utilizing a Pareto frontier. *Technometrics* 2011; **53**:353–365.
47. Xu H. An algorithm for constructing orthogonal and nearly-orthogonal arrays with mixed levels and small runs. *Technometrics* 2002; **44**:356–368.

Authors' biographies

C. Devon Lin is an Assistant Professor in Department of Mathematics and Statistics at Queen's University. Her research interests include design of experiments and design and analysis of computer experiments.

Christine M. Anderson-Cook is a Research Scientist at Los Alamos National Laboratory. Her research interests include reliability, design of experiments, multiple criteria optimization, and response surface methodology. She is a Fellow of the American Statistical Association and the American Society for Quality.

Michael S. Hamada is a Scientist at Los Alamos National Laboratory. His research interests include measurement systems, design and analysis of experiments, and reliability. He is a Fellow of the American Statistical Association.

Leslie M. Moore is a Guest Scientist at Los Alamos National Laboratory. She received her Ph.D. in Mathematics with concentration in Statistics from the University of Texas at Austin. Her research interests include statistical design of experiments for physical and computer experiments.

Randy R. Sitter was a Professor in Department of Statistics and Actuarial Science at Simon Fraser University. His research interests include statistical design and analysis of physical experiments and computer experiments, survey sampling, and industrial statistics.