Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

An Algorithm for Generating Good Mixed Level Factorial Designs

Ulrike Grömping[a,*] and Roberto Fontana[b]

[a] *Beuth University of Applied Sciences, Berlin, Germany*

[b] *Politecnico di Torino, Torino, Italy*

Abstract

An algorithm for the creation of mixed level arrays with generalized minimum aberration (GMA) is proposed. GMA mixed level arrays are particularly useful for experiments involving qualitative factors: for these, the number of factor levels is often a consequence of subject matter requirements, while a priori assumptions on a statistical model are not made, apart from assuming lower order effects to be more important than higher order effects. The proposed algorithm creates GMA arrays using mixed integer optimization with conic quadratic constraints. Fully achieving GMA is feasible for small problems; for larger problems, the optimization task is reduced to considering the confounding of low-order effects only. Lower bounds for the lowest-order confounding are provided (given the number of experimental runs). Where one of these bounds is actually attainable, the algorithm is often fast in providing an array which attains it. Examples illustrate the scope and usefulness of the algorithm, which is implemented in an R package, using one of two commercial optimizers.

Keywords: Experimental design; Orthogonal arrays; Generalized minimum aberration; Mixed integer optimization

1 Introduction

Factorial experiments are often run using orthogonal arrays. For example, engineers make ample use of the collection of arrays provided by the Japanese engineer Genichi Taguchi (see e.g. NIST Sematech 2016). Mixed level experiments, i.e. experiments for which not all factors have the same number of levels, are common in applications, especially if some factors are qualitative in nature. If a particular mixed level experiment is required, availability of a suitable array can be an issue. It is common to create a

* Correspondence to: Ulrike Grömping, Dep. II, Beuth University of Applied Sciences, Luxemburger Str. 10, D-13353 Berlin, Germany. Phone: +49 (0) 30 4504 5127. E-mail address: groemping@beuth-hochschule.de.

+ This paper also includes Supplementary material providing designs (72runsGOOD.rda, MIP12.rda, largedesignsTable5.rda) and instructions to using these as well as additional examples (SupplCSDA.pdf).

factorial design from a subset of the columns of a published array, and Grömping (2018c) discussed ways to improve this type of usage by optimizing the choice of columns.

Of course, optimization of column selection from an existing array cannot be better than the creation of a tailor-made optimized array for the task at hand. For experiments with some qualitative factors, one will often not have a particular model in mind, but will aim for the estimation of main effects, perhaps also of two-factor interactions, assuming that lower order effects are more important than higher order effects. For such a context, Fontana (2017) introduced an algorithm for the creation of an orthogonal array in a given number of runs that fulfills the quality criterion "generalized minimum aberration" (GMA, see Section 2.1); his formulation relied on the complex coding (Bailey 1982). For m factors, Fontana's algorithm used a sequence of m mixed integer quadratic optimization steps, with all but the first problems also having conic quadratic constraints. Because of the complex coding, the matrices for the target function and for the constraints of each step were constructed using the Cholesky decomposition of an NxN matrix, where N denotes the number of runs needed for a full factorial experiment in the factors to be investigated, i.e. is usually very large. The intriguing feature about the approach is its principal ability to *algorithmically* create an *optimized* array for a specific experimental situation with mixed numbers of levels that is at the same time *model-robust*. Unfortunately, the details of the algorithm's implementation were such that its application in practice would only work for very small data situations; we set out to improve that situation.

We modify and improve Fontana's proposal in the following ways: Based on Gr"omping (2018a), we substantially reduce the sizes of the matrices in the optimization problems. Furthermore, we reduce the number of optimization problems to be considered, by replacing the first few conic quadratic problems with a single linear one that enforces zero confounding for effects up to a specified order (Fontana alluded to such a possibility; its implementation in his algorithm would have required the use of strata according to Fontana 2013, because of using the complex coding). For the objective of the first actual minimization, we derive a lower bound, which, if attained, allows to finish this optimization step fast (see Sections 2.3 and 4.4). Furthermore, we implement a loop over optimizations of different problem representations which proves very successful in increasing the chance that the first optimization step finds an optimum. In addition, we choose a pragmatic approach which allows us to provide good designs for larger problems: due to the nature of the optimization problem to be solved by mixed integer optimization, establishing overall GMA is extremely time consuming or even impossible for many problems of relevant sizes. Therefore, we primarily aim for minimizing confounding of lowest-order effects. There still remain cases for which neither confirmation of an optimum nor further improvement is possible; for these, we consider it valuable to provide "improved" arrays, even if they are not necessarily optimal. All these – GMA designs, designs with optimized lowest-order confounding and "improved designs" – are what we call "good designs". We propose to use our algorithm for algorithmically obtaining good mixed level designs for situations for which a GMA array is not readily available.

Many specialized algorithms in the design community, e.g. the enumeration algorithms by Bulutoglu and

Margot (2008) for symmetric orthogonal arrays or by Schoen, Eendebak and Nguyen (2010) for general orthogonal arrays, are intended as a tool for design researchers, for enumerating all existing designs or for providing an authoritative list of designs that practitioners can turn to. Other algorithms, e.g. the Kobilinsky et al. (2017) algorithm, can be used in design creation of a (mixed level) factorial design for a specific experiment. Our algorithm is of that latter nature. It is more general than the Kobilinsky et al. (2017) algorithm, in that it incorporates (part of) the GMA quality criterion (see Section 2.1) into design creation; furthermore, the Kobilinsky et al. (2017) algorithm is restricted to the construction of arrays that can be obtained from design keys (using pseudofactors), whereas our algorithm can construct arrays from the larger class of *all* orthogonal arrays. On the other hand, the Kobilinsky et al. (2017) algorithm can account for randomization restrictions, which is not in the scope of our algorithm. Both algorithms have been implemented in an R package (Grömping 2018b, R package DoE.MIParray; Kobilinsky et al. (2018), R package planor).

This paper derives the algorithm and provides examples that evidence its usefulness. All calculations have been done using the afore-mentioned R package DoE.MIParray, which offers functions based on two different commercial solvers for mixed integer optimization problems (Mosek and Gurobi, as documented in Mosek ApS 2017 and Gurobi Optimization, Inc. 2017). Both vendors provide free academic licenses and R packages that support access from within R. Mixed integer optimization is an area of active research; thus, what seems currently extremely challenging may become easily doable with future improvements to mixed integer algorithms. It is therefore possible that the range of applicability for the algorithm will broaden over time. Nevertheless, mixed integer optimization remains an NP hard problem. We present more detail than Fontana (2017) on the algorithm's implementation in the two solvers; the purpose of presenting the specifics of inputs to the commercial solvers is to enable readers to create their own implementations in other professional solvers, e.g. CPLEX. Apart from giving a high-level overview, we do not discuss any specifics of how mixed integer optimization is implemented within the solvers we use; readers are referred to the literature, e.g. the very accessible modeling cookbook provided by Mosek Aps (2018) and references therein.

Section 2 introduces the necessary fundamentals and some basic results. Section 3 provides a motivating example that illustrates the type of design that benefits most from our algorithm, Section 4 describes the algorithm, and Section 5 applies it to the test cases of Fontana (2017) (5.1) and to further interesting mixed level requests for orthogonal arrays (5.2) or supersaturated strength 1 arrays (5.3). The discussion highlights the merits of our proposal and points out needs for further research.

2 Basics and Auxiliary Results

An array d for accommodating m factors in n experimental runs can be written as an $n \times m$ array of symbols. The jth factor has s_j levels, which are denoted as $1, \ldots, s_j$, and the n rows of the array d contain the factor level combinations for n experimental runs. An orthogonal array (OA) of strength t

Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

in n runs with m_1 factors at s_1 levels, ..., m_k factors at s_k levels is denoted as $\mathrm{OA}(n, s_1^{m_1} \dots s_k^{m_k}, t)$: for an OA of strength t, each t-tuple of factors has each level combination the same number of times. Usually, OA's with strength 2 or more are considered; we also consider OA's of strength 1 for which each factor has each of its levels the same number of times, but level combinations of pairs of factors need not be balanced. In statistical applications, one often considers the resolution R (denoted by a roman numeral), which is $t + 1$. An array is called "supersaturated", if there are fewer rows than needed for accomodating all main effects degrees of freedom. Supersaturated arrays can have at most strength 1, and we will restrict attention to arrays with at least strength 1.

We denote (column) vectors as bold face lower case letters, matrices as bold face capitals; $\mathbf{1}_n$ or $\mathbf{0}_n$ denote column vectors of n ones or zeroes, respectively, $\mathbf{I}_n$ denotes the n-dimensional identity matrix, $\mathbf{J}_n$ an $n \times n$ matrix of ones. Comparison for vectors (e.g. $\mathbf{r} \geq \mathbf{0}_N$) is to be understood element wise. $\otimes$ denotes the Kronecker product, $\star$ the element wise (i.e. Hadamard) product, and the superscript $\top$ denotes the transpose.

The first sub section presents known results and notation on GMA and related topics. Sub sections 2.2 and 2.3 derive new and useful auxiliary results which support an efficient implementation of our algorithm. Sub section 2.4 presents known facts on quadratic cones and mixed integer optimization.

2.1 GMA and related metrics, and counting vectors

For the considerations to follow, it is helpful to represent the $n \times m$ array d via a counting vector $\mathbf{r}$. The idea is simple: a full factorial array in the m factors would have $N = \prod_{j=1}^{m} s_j$ runs. We define $\mathcal{D}$ as this full factorial in lexicographic order (i.e. levels of the jth factor from 1 to s_j, leftmost factor changing levels most slowly). We can then represent an $n \times m$ array d by the $N \times 1$ vector $\mathbf{r}$ that contains for each row of $\mathcal{D}$ the frequency with which it is contained as a row in d. This vector is called the counting vector. Of course, the sum of all elements of $\mathbf{r}$ is n and $\mathbf{r} \geq \mathbf{0}_N$.

The quality criterion "generalized minimum aberration" (GMA) was introduced by Xu and Wu (2001) and is based on the generalized word length pattern (GWLP), which measures the amount of confounding with the overall mean for each effect order (0=intercept=overall mean, 1=main effects, 2=two-factor interactions, ...). The GWLP is calculated from model matrices in "normalized orthogonal coding":

Definition 1 (normalized orthogonal coding)**.** Let $\mathbf{M} = (\mathbf{M}_0 \vdots \mathbf{M}_1 \vdots \dots \vdots \mathbf{M}_m)$ denote the $N \times N$ model matrix of the full model for the unreplicated full factorial design $\mathcal{D}$, with $\mathbf{M}_j$ denoting the $N \times df(j)$ matrix of interaction effect columns for all j-factor interactions. Then $\mathbf{M}$ is said to have normalized orthogonal coding, if

(i) $\mathbf{M}_0 = \mathbf{1}_N$,

(ii) all columns of $\mathbf{M}_1$ (main effects columns) have mean 0, squared Euclidean norm N and are orthogonal to each other,

(iii) and the $df(j)$ columns of $\mathbf{M}_j$, $j > 1$, are obtained as the products of j columns from $\mathbf{M}_1$ that refer to j distinct factors.

Definition 1 (i) and (iii) correspond to the usual way of obtaining model matrix columns for the intercept and for interaction effects. The critical step is thus the choice of a coding for the main effects; examples of normalized orthogonal codings for 2-level, 3-level and 4-level main effects factors are given in Table 1. Note that the complex coding obtained from the sth roots of the unity is another example of normalized orthogonal coding. Furthermore, note that Definition 1 implies that all columns of the matrix $\mathbf{M}$ have mean zero and squared norm N and are orthogonal to each other.

For actual designs (called fractions $\mathcal{F}$ in the following), the model matrix $\mathbf{M}_\mathcal{F} = (\mathbf{1}_n \vdots \mathbf{M}_{\mathcal{F};1} \vdots \ldots \vdots \mathbf{M}_{\mathcal{F};m})$ for the full model is obtained from $\mathbf{M}$ by omitting runs not present in the fraction and duplicating runs occurring multiple times, if applicable. $\mathbf{M}_\mathcal{F}$ has normalized orthogonal coding if the $\mathbf{M}$ from which it has been obtained complies with Definition 1. The GWLP is denoted as $(A_0, A_1, \ldots, A_m)$, with $A_0 = 1$ (confounding of the overall mean with itself). A_j is called the number of (generalized) "words" of length j, and it is calculated as the sum of squared column averages over all $df(j)$ columns of $\mathbf{M}_{\mathcal{F};j}$, where $\mathbf{M}_\mathcal{F}$ has normalized orthogonal coding. The strength t of an array is any positive integer for which $A_1 = \cdots = A_t = 0$; one usually identifies t with the maximum possible strength; the resolution of an array is the integer R such that $A_1 = \cdots = A_{R-1} = 0, A_R > 0$; we thus have $R = t + 1$, as was mentioned before. GMA consists in minimizing A_1, A_2, A_3, ... in turn, i.e., one first maximizes the resolution, and then minimizes the number of shortest words A_R, followed by the number of second shortest words A_{R+1}, and so forth.

A_R is a summary metric: it sums the confounding over all R factor sets; Grömping and Xu (2014) introduced a mixed level version of "generalized resolution" (GR), which measures the distance of the worst case R factor set from complete confounding. For GR $= R$, there is at least one R factor set, in which at least one factor's main effect is completely confounded by the interaction of the other $R - 1$ factors. GR $> R$ is thus higly desirable, and is particularly important for resolution II designs, for which complete confounding confounds main effects with each other.

Using the counting vector $\mathbf{r}$, we can write $A_j = \frac{1}{n^2}\mathbf{1}_n^\top \mathbf{M}_{\mathcal{F};j}\mathbf{M}_{\mathcal{F};j}^\top \mathbf{1}_n = \frac{1}{n^2}\mathbf{r}^\top \mathbf{M}_j \mathbf{M}_j^\top \mathbf{r}$. Thus, minimizing A_j is equivalent to minimizing the quadratic form $\mathbf{r}^\top \mathbf{M}_j \mathbf{M}_j^\top \mathbf{r}$ w.r.t. the choice of a non-negative $N \times 1$ integer counting vector $\mathbf{r}$ with sum n. The matrix $\mathbf{H}_j = \mathbf{M}_j \mathbf{M}_j^\top$ has some useful properties which help to make the optimization feasible. Subsection 2.2 derives these.

For strength 1 OAs, i.e. for balanced possibly supersaturated arrays, it is unusual to consider their quality in terms of A_2. For 2-level designs, the typical metric is $E(s^2)$, which is equivalent to A_2 according to Xu (2015); for mixed level designs, two frequently used metrics are $E(\chi^2)$ and $E(f_{NOD})$. According to Grömping (2017), $E(\chi^2) = nA_2/\binom{m}{2}$, so that optimizing A_2 is equivalent to optimizing $E(\chi^2)$ (see also Section 2.3).

Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

Table 1: Contrast matrices for $s = 2, 3, 4$

$s=2$	$s=3$		$s=4$		
-1	$-\sqrt{3/2}$	$-\sqrt{1/2}$	$-\sqrt{2}$	$-\sqrt{2/3}$	$-\sqrt{1/3}$
1	$\sqrt{3/2}$	$-\sqrt{1/2}$	$\sqrt{2}$	$-\sqrt{2/3}$	$-\sqrt{1/3}$
	0	$\sqrt{2}$	0	$\sqrt{8/3}$	$-\sqrt{1/3}$
			0	0	$\sqrt{3}$

2.2 Properties of $\mathbf{H}_j$

Before looking at $\mathbf{H}_j$ itself, we consider cross products of normalized orthogonal contrast matrices. Let $\mathbf{C}_s$ denote the $s \times (s-1)$ main effect contrast matrix for an s-level factor in normalized orthogonal coding; the matrices in Table 1 are examples of $\mathbf{C}_2$, $\mathbf{C}_3$ and $\mathbf{C}_4$. Grömping (2018a) showed that outer products of effect model matrices are invariant to the choice of normalized orthogonal coding. This applies in particular also to the matrices $\mathbf{C}_s$ themselves, which are effect model matrices in a simple one-factor model with s runs.

Lemma 1. *Let $\mathbf{C}_s$ denote the contrast matrix for an s-level effect in normalized orthogonal coding. Then $\mathbf{C}_s\mathbf{C}_s^\top = s\mathbf{I}_s - \mathbf{J}_s$.*

Proof. The proof is by induction. Because of the coding invariance of the outer product, we can use any particular normalized orthogonal coding and will use the normalized Helmert coding for which the first few contrast matrices are given in Table 1. For 2-level factors, the assertion is trivial. Assume that the assertion holds for an s-level factor. The contrast matrix $\mathbf{C}_{s+1}$ can be written as

$$\mathbf{C}_{s+1} = \left(\begin{array}{c|c} \sqrt{\frac{s+1}{s}}\mathbf{C}_s & -\sqrt{\frac{1}{s}}\mathbf{1}_s \\ \hdashline \mathbf{0}_{s-1}^\top & \sqrt{s} \end{array}\right).$$

With straightforward calculations, this implies the stated form for the outer product matrix $\mathbf{C}_{s+1}\mathbf{C}_{s+1}^\top$. □

$\mathbf{H}_j = \mathbf{M}_j\mathbf{M}_j^\top$ is the sum of the outer products of the $df(j)$ individual columns of $\mathbf{M}_j$, but can also be written as a sum of outer product matrices over all $\binom{m}{j}$ j-factor interactions, i.e. as $\sum_{S\subseteq\{1,\ldots,m\},|S|=j} \mathbf{X}_{\mathcal{I}(S)}\mathbf{X}_{\mathcal{I}(S)}^\top$, where $\mathbf{X}_{\mathcal{I}(S)}$ denotes the model matrix from the interaction among the factors in the set $S = \{i_1, \ldots, i_j\}$ for the unreplicated full factorial array $\mathcal{D}$. The following lemma states the resulting properties of the $\mathbf{H}_j$, which are a direct consequence of the lexicographic order of the full model matrix and Grömping's (2018a) results on the nature of the model matrices.

Lemma 2. *The matrix $\mathbf{H}_j = \mathbf{M}_j\mathbf{M}_j^\top$ has the following properties:*

(i) $\mathbf{H}_j$ *can be written as*

$$\mathbf{H}_j = \sum_{\mathcal{I}(S):|S|=j} \bigotimes_{i=1}^{m} \left(s_i\mathbf{I}_{s_i} - \mathbf{J}_{s_i}\right)^{i\in\mathcal{I}(S)},$$

Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

where the exponents in the Kronecker product are logical values interpreted as zeroes and ones and are applied element wise, i.e. exponentiation with 0 keeps the dimensions and makes all elements equal to one, while exponentiation with 1 keeps the unchanged matrix.

(ii) All elements of $\mathbf{H}_j$ *are integers.*

(iii) $\mathbf{H}_j$ *is positive semidefinite with rank df(j).*

(iv) All elements of $\mathbf{H}_j$ *sum to zero.*

Proof. The proof is sketched only. Part (i) follows from

$$\mathbf{X}_{\mathcal{I}(S)} = \bigotimes_{i=1}^{m} \begin{cases} \mathbf{1}_{s_i} & i \notin S \\ \mathbf{C}_{s_i} & i \in S \end{cases}$$

and Lemma 1. Lemma 2 (ii) is a consequence of (i). Part (iii) follows from $\mathbf{H}_j$ being a sum of Hadamard products of non-negative definite matrices (according to Grömping 2018a), and part (iv) holds, because the full factorial array, represented by $\mathbf{r} = \mathbf{1}_N$, has $A_j = 0$ for all $j > 0$. □

The algorithm will make use of the factorization $\mathbf{H}_j = \mathbf{M}_j\mathbf{M}_j^\top$ with the $N \times df(j)$ matrix $\mathbf{M}_j$. This factorization brings a substantial advantage versus the proposal by Fontana (2017), who used the Cholesky decomposition $\mathbf{H}_j = \mathbf{U}_j\mathbf{U}_j^\top$ with square matrices $\mathbf{U}_j$ instead: since the number of additional variables needed in conic quadratic constraints (see Section 4) depends on the number of columns of $\mathbf{M}_j$ or $\mathbf{U}_j$, respectively, our formulation requires fewer variables and thus brings larger problems into reach. Nevertheless, the size N of the full factorial is one of the limiting elements for usability of the algorithm.

2.3 Lower bounds for A_R and sum of the A_j

Grömping and Xu (2014) reported a lower bound for A_R in R factor arrays of resolution R (their Theorem 5); for m factor arrays with $m > R$, a lower bound can be derived by summing the lower bounds over all R factor subsets of the m factors. The algorithm discussed later in this paper can make use of a lower bound for n^2A_R; this is provided in the following two propositions.

Proposition 1. Consider an array d in n runs and m factors with resolution R (i.e., with strength $t = R - 1$). Then,

$A_R(d) \cdot n^2 \geq \sum_{S\subseteq\{1,\ldots,m\},|S|=R}(\prod_{i\in S} s_i - r_S) \cdot r_S$, with r_S being the remainder when dividing n by $\prod_{i\in S} s_i$.

For example, if an OA(18, $2^1 3^3$, 2) is sought, there are three 3-factor sets S with one 2-level factor and two 3-level factors each ($r_S = 0$) and one 3-factor set S with three 3-level factors ($r_S = 18$). With Proposition 1, $A_3 \cdot 18^2 \geq 3 \cdot (18 - 0) \cdot 0 + (27 - 18) \cdot 18 = 9 \cdot 18$, which implies $A_3 \geq 0.5$.

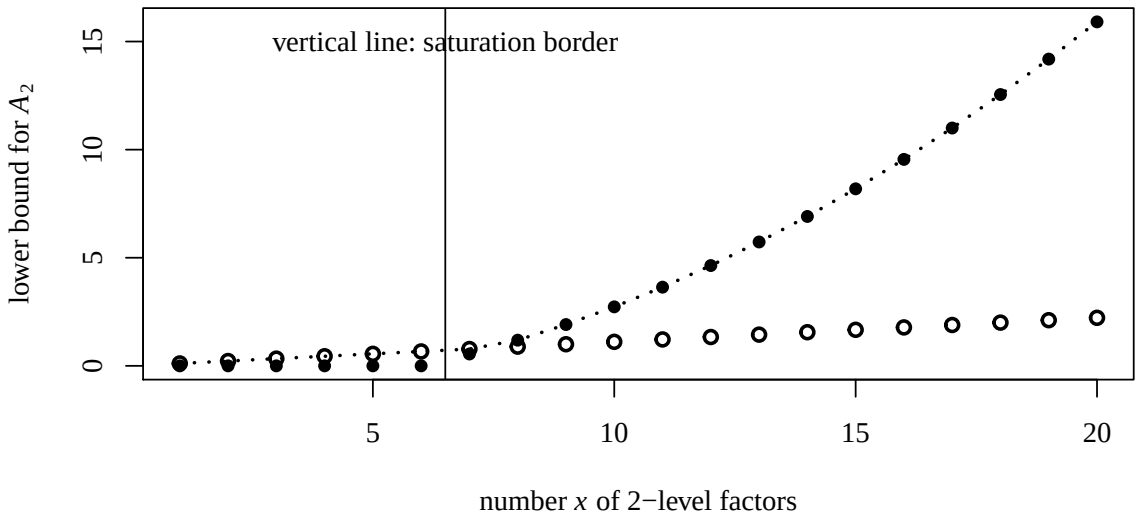

Figure 1: Bounds of Propositions 1 (open symbol) and 2 (closed symbol) for OA(12, $2^a3^14^1$, 1)

Following Grömping and Xu (2014), the bound of Proposition 1 is fulfilled, if and only if all R factor sets have weak strength R, i.e. have resolution R and do not have duplicate rows. This is most often the case for relatively small mixed level arrays, for which the algorithm is most useful. For symmetric s-level arrays, the bound is zero whenever n is a multiple of s^R, because $r_S = 0$ for such situations.

For balanced supersaturated arrays, which have resolution II, Ai, Fang and He (2007) and Liu and Lin (2009, their Lemma 2) published equivalent lower bounds for the quality criterion "expected χ^2 value" or "sum of χ^2 values" (related to each other by the factor $\binom{m}{2}$ for the number of pairs) that are closely related to A_2: according to Grömping (2017), n times the A_2 contribution of a pair of factors equals the χ^2 contribution of the pair. Thus, these lower bounds can be easily adapted for A_2 (multipliers $1/(n\binom{m}{2})$ or $1/n$, respectively), see the following proposition. In general, the lower bound derived from the Ai, Fang and He (2007) or Liu and Lin (2009) bound is sharper for supersaturated arrays, while the bound from Proposition 1 is sharper for arrays that are not saturated; this is illustrated by Figure 1 for 12-run arrays with one 3-level factor, one 4-level factor and x 2-level factors (for $x < 6$, negative calculation results for the bound from Proposition 2 have been replaced by the trivial lower bound 0).

Proposition 2. Consider an array d in n runs and m factors with resolution 2 (i.e., with strength 1), with factor i at s_i levels, $i = 1, \ldots, m$. Then,

$$A_2(d) \cdot n^2 \geq \frac{n^2}{2(n-1)} \Big(\big(\textstyle\sum_{i=1}^m s_i\big)^2 - (n - 1 + 2m) \big(\textstyle\sum_{i=1}^m s_i\big) + m(m + n - 1) \Big).$$

Note that n^2A_2 is always integer. Thus, in order for the bound of Proposition 2 to be sharp, it must be integer, which is often not the case; thus, it can be slightly sharpened for practical use.

Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

The maximum of the bounds resulting from Propositions 1 and 2 can be used in the optimization process; since mixed integer optimization can be extremely slow to confirm optimality, its incorporation can be very helpful for situations for which it is in fact attained, because optimality is then known and does not need to be confirmed by reducing the gap to zero in the mixed integer algorithm (see Section 4).

In addition to the lower bounds for A_R, for arrays with distinct rows, i.e. counting vectors $\mathbf{r}$ with binary elements, a simple formula for the sum of the A_j can be given:

Proposition 3. Consider m factors with $s_1, \ldots, s_m$ levels, for which $N = \prod_{j=1}^{m} s_j$ is the size of an unreplicated full factorial array $\mathcal{D}$. For any unreplicated n run array d in these m factors, the sum of the GWLP entries is $\sum_{j=0}^{m} A_j(d) = N/n$.

In words, Proposition 3 states that the sum of the A_j is the inverse proportion of the degrees of freedom needed for a full model (equal to the run size of the full factorial) that can be accomodated in an n run array. For regular symmetrical s-level arrays, Proposition 3 follows from the number of generators needed, noting that each generator contributes $s-1$ to the sum (i.e. the number of words has to be multiplied with $s-1$ for obtaining the entries of the GWLP). For general symmetrical arrays, Proposition 3 directly follows from Xu and Wu's equation (8) with $j=0$, and noting that their $B_0(d) = 1$ if and only if array d does not have repeated rows and that their $B'_j(d) = A_j(d)$. For mixed level arrays, the analogous reasoning is not explicated in Xu and Wu (2001), but does also apply (personal communication with Hongquan Xu). For obtaining GMA arrays, Proposition 3 implies that it suffices to optimize $A_1, \ldots, A_{m-1}$, because $A_m = N/n - 1 - \sum_{i=1}^{m-1} A_i$, provided d has distinct rows.

2.4 Quadratic cones and Mixed Integer Optimization

A cone is a set C for which $a \in C \implies \forall \lambda \geq 0 : \lambda a \in C$. A quadratic cone can be defined as $\mathcal{Q}^k = \{(x_1, x_2, \ldots, x_k) : x_1 \geq \sqrt{x_2^2 + \cdots + x_k^2}\}$, i.e. consists of vectors whose first element is at least the Euclidean norm of the vector of the remaining components. It is obvious that $\mathcal{Q}^k$ is a cone. We will point out that the problem of minimizing A_j can be written as minimizing the (squared) first coordinate of a quadratic cone, given a set of equality constraints on the other coordinates. For casting this problem into the format required by Mosek or Gurobi, it is necessary to use a more general form: a rotated quadratic cone is defined as $\mathcal{Q}_r^k = \{(x_1, x_2, \ldots, x_k) : x_1, x_2 \geq 0 \text{ and } 2x_1x_2 \geq x_3^2 + \cdots + x_k^2\}$. $\mathcal{Q}_r^k$ has been written in the Mosek version of the definition (see Mosek ApS 2014); Gurobi omits the "2" on the left-hand side; this paper uses the Mosek version throughout. Like $\mathcal{Q}^k$, $\mathcal{Q}_r^k$ is obviously a cone. A $k+1$-dimensional *rotated* quadratic cone with its second coordinate fixed at $1/2$ (Mosek; for Gurobi, one would fix it at 1) can be used to represent a k-dimensional *unrotated* quadratic cone, using the equivalence relation $(\sqrt{x_1}, x_2, \ldots, x_k) \in \mathcal{Q}^k \Leftrightarrow (x_1, 1/2, x_2, \ldots, x_k) \in \mathcal{Q}_r^{k+1}$. Thus, the quadratic cones needed for our algorithm can be represented in the general rotated quadratic cone framework employed by Mosek and Gurobi, respectively.

The documentations for Gurobi and Mosek (Gurobi Optimization, Inc. 2017 and Mosek ApS 2017) describe

general mixed integer optimization strategies; specific details are partly hidden to the public. In short, strategies combine solving a relaxed problem without integer constraints and then trying to find integer solutions that are (almost) as good; heuristics are used in addition to simplex methods and branch and bound / branch and cut strategies, and optimality is proven, if the optimum of the relaxed problem equals the integer optimum. The discrepancy between relaxed and integer problem is called the "gap", and it is usually considered in relative terms as

$$\text{gap} = \frac{\text{current integer minimum} - \text{current relaxed minimum}}{\text{current integer minimum}}.$$

The current relaxed minimum is (of course) a lower bound for the integer minimum; providing the algorithm with a known more aggressive lower bound helps to prove optimality; with Gurobi and Mosek, the gap is nevertheless calculated based on the current relaxed minimum rather than a user-provided lower bound. Optimality is considered confirmed, when the gap becomes zero or a user-provided lower bound is reached. Thus, mixed integer optimization has two tasks: driving the current minimum as far down as possible and proving its optimality by reducing the gap. In many examples, reaching the actual optimum happens much faster than proving its optimality.

Our objective function is quadratic in the integer vector $\mathbf{r}$. Today's optimization software transforms minimization of a quadratic objective into minimization of a linear objective under a conic quadratic constraint. This is for example described in Mosek ApS (2018), and is the reason why we introduced quadratic cones.

3 A motivating example

Larger arrays with strength 2 and far from saturating main effects df cannot be found easily in published tables; for example, an optimized OA(72, $2^4 3^2 4^1$, 2) is neither found on the Eendebak and Schoen (2010) website (which covers strength 2 arrays with up to 70 runs only) nor on the Kuhfeld (2009) website; the latter provides many instances of 72 run arrays with many main effects df. Grömping (2018c) reported on the creation of an OA(72, $2^4 3^2 4^1$, 2) by optimizing column selection from an OA(72, $2^{43} 3^8 4^1 6^1$, 2) taken from the Kuhfeld (2009) collection; this array reached $(A_3, A_4) = (0.451, 3.247)$ and was used for a biotechnological experiment (see Vasilev et al. 2014).

With our algorithm, a design with the globally optimal $A_3 = 0.074$ (as confirmed by the lower bound of Proposition 1) was found. Situations like this, where a mixed level design is sought for which the next higher strength is not quite possible but A_R is small, benefit the most from using our algorithm. The main purpose of our algorithm is to enable researchers to produce a tailor-made design on demand in such situations. Note that an attempt to improve the design's A_4 was unsuccessful, and neither was optimality of A_4 confirmed, so that we do not know whether the design has GMA or whether its A_4 could be improved; this is typical for attempts on improving A_{R+1}, unless the design is very small. Note that

Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

the design of this brief section is part of the 72 run series worked out in the "Examples" section.

4 The algorithm

The purpose of this section is to explain the use of Mosek or Gurobi for finding a GMA design or –more often –for finding a resolution R design and optimizing A_R. The integer optimization itself is left to these tools and not described here in any detail, as it would be unwise to program this instead of relying on the highly sophisticated existing tools.

4.1 Mixed integer optimization for our problem

The objective $n^2 A_j = \mathbf{r}^\top \mathbf{H}_j \mathbf{r}$ (to be minimized) is quadratic in $\mathbf{r}$. The following section details the optimization problems to be solved for optimizing the GWLP. We build on Section 2.4, where quadratic cones and the main concepts of mixed integer optimization were outlined.

Both Gurobi and Mosek offer APIs for R, which are provided in the R packages Rmosek and gurobi, respectively. With these, optimization problems can be handed to the optimizers directly from R, without having to handle the optimizers' interfaces. Both Gurobi and Mosek are – to a certain extent – run-to-run deterministic, i.e. do not contain random decision elements. Differences between runs can nevertheless occur

- whenever a time limit is involved, due to resource availability on the computer,
- if different numbers of cores are used (we use two cores throughout),
- between machines with different abilities (e.g. if resource-intensive heuristics work better on the more powerful machine).

Both Mosek and Gurobi offer options for controlling the solvers' behavior in terms of presolve activities, the intensity of using certain heuristics, and tolerances. Gurobi additionally offers two options for strategic decisions, called `MIQCPMethod` and `MIPFocus`. For both, defaults are used, but can be overwritten by the user. For our problem, `MIQCPMethod=0` is often an important option to set (and is the default in R package DoE.MIParray). The parameter `MIPFocus` controls whether Gurobi should focus more on finding new feasible solutions or on improving the lower bound; within package DoE.MIParray, this parameter defaults to focussing on finding new feasible solutions. Nevertheless, in many examples considered, Mosek, which does not offer this kind of choice to the user, performs better in finding a good solution fast. All documentation is freely available on the web, and a detailed discussion of the optimizers' possibilities would go beyond the scope of this article. In principle, the instructions in this paper should enable interested readers to implement our algorithm in other optimizers that offer convex integer optimization under conic quadratic constraints.

We want to remark on timings, which we will provide in the Examples section. Timings are not about

comparing the two optimizers; comparison might be unfair, because we may not have used the best possible settings of optimizer options. On the other hand, an optimizer that does not require the user to specify many options and still performs well has an advantage in our point of view. Although this paper is not about comparing the two optimizers, we provide a few cautious comments on their performance. In our examples, Mosek was often faster than Gurobi in finding a good design; on the other hand, Gurobi was sometimes faster than Mosek in proving non-existence of a design. For both, the performance sometimes strongly depended on the order in which the factors were provided to the optimizers, even though this order does of course not change the structure of the problem. It does, however, change the order in which the algorithms search the huge search space and is therefore consequential. Unfortunately, there seem to be no general rules that govern whether an order is fast or slow.

4.2 The optimization problems

For achieving GMA, one has to successively request minimization of A_1, A_2, A_3, and so forth. Since each minimization may take a lot of effort, avoidance of minimization steps is a great time saver. We therefore implement a resolution option: if resolution R_0 is requested, the algorithm searches for a feasible counting vector that respects the requested resolution. This is an integer-constrained linear optimization problem with constant objective function, which can usually be solved much faster than the conic quadratic optimization problems of later steps (see problem (1) below). The potential for this type of initial step was already mentioned, but not implemented, in Fontana (2017). Subsequently, A_{R_0} is minimized, using optimization problem (2L), which is equivalent to optimization problem (2Q).

(1) $\min 0$

subject to

$\mathbf{1}_N^\top \mathbf{r} = n,$

$\mathbf{M}_{<R_0}^\top \mathbf{r} = \mathbf{0}_{df_{<R_0}},$

$\mathbf{r} \geq \mathbf{0}_N,$

$\mathbf{r}$ integer.

(2Q) $\min \mathbf{r}^\top \mathbf{H}_{R_0} \mathbf{r}$

subject to

$\mathbf{1}_N^\top \mathbf{r} = n,$

$\mathbf{M}_{<R_0}^\top \mathbf{r} = \mathbf{0}_{df_{<R_0}},$

$\mathbf{r} \geq \mathbf{0}_N,$

$\mathbf{r}$ integer.

(2L) $\min t_1$

subject to

$\mathbf{1}_N^\top \mathbf{r} = n,$

$\mathbf{M}_{<R_0}^\top \mathbf{r} = \mathbf{0}_{df_{<R_0}},$

$\mathbf{M}_{R_0}^\top \mathbf{r} - \mathbf{y} = \mathbf{0}_{df(R_0)},$

$\mathbf{r} \geq \mathbf{0}_N,$

$\mathbf{r}$ and t_1 integer,

$t_2 = 0.5,$

$(t_1, t_2, y_1, \ldots, y_{df(R_0)})^\top \in \mathcal{Q}_r^{df(R_0)+2}.$

In the optimization problems, we denote by $\mathbf{M}_{<R_0}$ the model matrix for the main effects and (for $R_0 > 2$) all interactions up to degree $R_0 - 1$, and by $df_{<R_0}$ the number of columns of this matrix. The run size constraint is written in terms of the summation vector $\mathbf{1}_N$, which is identical to the model matrix $\mathbf{M}_0$, which is *not* contained in $\mathbf{M}_{<R_0}$. Problem (2L) recasts the quadratic problem (2Q) as a linear problem with conic quadratic constraint. All three problems require $\mathbf{r}$ to be integer; in many cases, $\mathbf{r}$ can (and

should) instead be constrained to be binary, which requests a design with distinct rows.

Equivalence of the problems (2Q) and (2L) can be easily verified:

a. $\mathbf{H}_{R_0} = \mathbf{M}_{R_0}\mathbf{M}_{R_0}^\top$, with the $N \times df(R_0)$ matrix $\mathbf{M}_{R_0}$. With the $df(R_0) \times 1$ vector $\mathbf{y} = \mathbf{M}_{R_0}^\top \mathbf{r}$, $\mathbf{r}^\top \mathbf{H}_{R_0} \mathbf{r} = \mathbf{y}^\top \mathbf{y} = \sum_{i=1}^{df(R_0)} y_i^2$.
b. The quadratic objective function is linearized by adding a new variable t_1 which becomes the objective function: Minimizing t_1, and at the same time constraining it by $t_1 \geq \mathbf{y}^\top \mathbf{y}$ (expressed as $(\sqrt{t_1}, y_1, \ldots, y_{df(R_0)}) \in \mathcal{Q}^{df(R_0)+1}$), will eventually yield the minimum possible $\mathbf{y}^\top \mathbf{y}$ as the objective value.
c. As the optimizers use *rotated* quadratic cones, the unrotated quadratic cone from b is embedded into this more general notation by introducing the additional variable t_2 with an appropriate equality constraint (see also Section 2.4), chosen as 0.5 for Mosek's approach.

If A_{R_0+1} is subsequently minimized, problem (2L) must be appropriately extended: it retains all constraints and adds a suitable optimality constraint, e.g. $t_1 \leq t_{1,min} + 0.3$ (where the "+0.3" avoids numerical problems and is not harmful, because t_1 is integer). Its quadratic objective function is linearized in the same way as done from problem (2Q) to problem (2L), i.e. $df(R_0+1)+2$ additional variables, $df(R_0+1)+1$ equality constraints and a conic quadratic constraint have to be added. Analogous extensions apply for further problems concerning the minimization of A_j with $j > R_0 + 1$, where applicable.

4.3 The overall algorithm

First, one decides on a suitable R_0, i.e. a resolution to be requested, and on a maximum word length to be considered, denoted as k_{max}. For designs with distinct runs, $k_{max} \leq m - 1$ suffices for obtaining a GMA design (according to Proposition 3). It will often be best to choose $k_{max} = R_0$ or at most $k_{max} = R_0 + 1$.

The algorithm has the following high-level steps:

1. Solve problem (1), i.e. obtain a vector $\mathbf{r}$ that satisfies the requested resolution R_0.
   If a solution is found, proceed to Step 2.
   If infeasibility is detected, modify the problem, e.g. by removing a request for binary elements of $\mathbf{r}$, by reducing the requested resolution, or by increasing n.
2. Solve problem (2Q) in the representation (2L), i.e. minimize A_{R_0}, subject to the resolution constraints, taking the solution of Step 1 as the starting value. Optimality is ascertained
   - if the lower bound of Section 2.3 is reached
   - or if the gap is reduced to zero.

   If optimality is confirmed:
   - If $k_{max} = R_0$, return the optimum solution.
   - If $k_{max} > R_0$ and the optimum is zero, increase R_0 to $R_0 + 1$ and rerun Step 2, using the solution of the previous run as the starting value.

Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

- If $k_{max} > R_0$ and the optimum is positive, proceed to Step 3.

If optimality is not confirmed within reasonable time (e.g. specified to the commercial solver as a time limit), return the best attained solution as a potential "good" design (see also Sections 4.4 and 5).

3. for $k_{max} > R_0$, and $A_{j-1} \geq R_0$ already minimized, minimize $A_j, R_0 < j \leq k_{max}$ (quadratic objective function), subject to all prior constraints obtained for $A_{j'}, R_0 \leq j' < j$, using the solution of the minimization of A_{j-1} as the starting value. Repeat this step as often as necessary. (The optimization problems for this step have not been explicitly stated, but their successive creation has been described in the previous section.)

Step 1 is often fast. Step 2 can be fast, if the lower bound is attained; otherwise, confirmation of optimality is successful for small problems only (see examples in Section 5.1 and 5.2.1). For such problems, it may also be feasible to enter Step 3, i.e. to optimize A_j values with $j > R_0$. If Step 2 does not yield a confirmed optimum, it is recommended to run it with suitable verbosity as long as the objective function improves for obtaining a "good" design, even if it is not necessarily optimum (see also Section 4.4).

4.4 Practical aspects

The algorithm is implemented in the R package DoE.MIParray; depending on the choice of k_{max}, it optimizes low order confounding only or (with $k_{max} = m-1$ (or m)) can be used to ensure GMA. However, there are resource-driven practical problems. Running optimization without explicit time constraints will often run for a very long time, with an eventual interrupt by the user, risking to loose results. Running with a time constraint guarantees that the result obtained will be stored, but unfortunately hot-starting is of limited value: one can use the latest solution as a starting value, but it is not possible to preserve the branch and bound interim results, which will have to be re-established, when re-starting the algorithm.

In many applications, obtaining GMA for all word lengths will not be feasible. Generally, it should be considered more important to optimize shorter word lengths, and, in particular, to achieve the best possible resolution. It can be recommended to start with an aggressive guess of the resolution, since infeasibility in Step 1 of the algorithm is often detected very fast, e.g. because one of the known rules for the existence of arrays is violated, which will be automatically detected. There are also other cases, however: for example, there is no OA(54, 3^6, 3) (see e.g. the website by Eendebak and Schoen 2010), although none of the feasibility bounds is violated; the infeasibility of this array is not established fast by either Mosek or Gurobi.

After establishing resolution, A_R is to be optimized; if this is successful, optimization of A_{R+1} is of interest; otherwise, one would rather further improve A_R. Whenever the lower bound according to Propositions 1 or 2 is reached (applicable for A_R only), one can safely switch to the next word length; for other cases, it is often not easy or even impossible to achieve the conviction that an optimum has been reached. It should usually be sufficient for practical purposes to optimize A_R and A_{R+1} (provided R is

Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

the resolution), or A_R only for the more difficult cases. If even optimality of A_R cannot be reached or confirmed, an array constructed by the algorithm may still be useful. Whenever a confirmed GMA array cannot be achieved, the optimization result should be compared to arrays obtained from other approaches, e.g. the column selection optimization implemented in R package DoE.base (Grömping 2018c).

Clearly, it is desirable to have an array consist of distinct runs only. Thus, it is worth a try to request binary instead of integer elements for the vector $\mathbf{r}$. Such a request, if feasible, should implicitly lead to better optima, because GMA arrays have unreplicated runs, where possible. Therefore, R package DoE.MIParray uses a restriction of $\mathbf{r}$ to binary as the default and allows relaxation to general integer as an option. Sometimes, the optimum array with distinct runs is nevertheless more easy to find by initially permitting repeated runs, i.e. general integer elements of $\mathbf{r}$. Playing with optimizers' options can also sometimes help to solve difficult cases, particularly with Gurobi.

Although the optimization problem (of course) remains the same regardless of the order in which the variables are entered, optimization speed and the quality of the improvement reached by an optimizer within reasonable time does sometimes (strongly) depend on that order; for example, the design obtained in Section 3 was obtained fast for two of the possible 105 level orders only; in this case (like in many other cases), optimization problem (1) is solved very fast for all level orders, and optimization problem (2L) leads to fast improvements of the objective function (which equals $72^2 A_3$), while the quality of the objective reached in short time strongly depends on the order of entering the variables. Using Mosek, entering the level numbers in the order 2,2,2,2,3,4,3 (or an alternative order reported later in Table 4), the algorithm reached the globally optimal $A_3 = 0.074$ (as confirmed by the lower bound of Proposition 1) within less than a minute. Many other orders of level numbers led to A_3 values that were much worse than this optimum and at least slightly worse than that of the design used in Vasilev at al. (2014). Because of the strong dependence on level orders, combined with the algorithm's tendency to improve the objective value very fast in many cases, the R package DoE.MIParray offers functions `mosek_MIPsearch` and `gurobi_MIPsearch` for looping through all different level orders, optimizing each for a specified time (default 60s). This search functionality has been used for several situations in the Examples section.

5 Examples

The calculations were done using two threads on a Windows machine with i7 processor with four cores and 32GB RAM, using the R package DoE.MIParray with R 3.4.1, Mosek version 8.1 and Gurobi version 8.0.1.

5.1 Test cases from Fontana (2017)

We start by addressing the test cases investigated in Fontana (2017). First, we consider accommodating five 2-level factors in 4 (not in Fontana 2017), 6, 8, 10, 12, 14 or 16 runs. We ran the algorithm

Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

Table 2: Run times (s) for finding the GMA array for five 2-level factors

n	R	A_R	bound	Fontana 2017	Mosek all	Gurobi all	Mosek from A_R	Gurobi from A_R
4	II	2.000	1.688		2.88	2.07	1.77	2.20
6	II	1.111	1.111	28	10.64	19.17	10.47	19.14
8	III	2.000	0.000	8	0.60	0.51	0.45	0.44
10	II	0.400	0.400	96	0.70	1.87	0.62	1.72
12	III	1.111	1.111	9	0.78	0.94	0.64	0.80
14	II	0.204	0.204	3415	1.34	4.15	1.28	3.93
16	V	1.000	1.000	29	0.32	0.27	0.14	0.16

from scratch without using any prior knowledge (i.e. optimizing A_1, A_2, and so forth, up to A_5, as proposed in Fontana 2017), or we requested each array to have the correct resolution R, so that only A_R to A_5 needed to be optimized (we ignored the fact that optimization of A_5 can be skipped due to Proposition 3). For both these approaches, we applied the default settings in the R package DoE.MIParray, using both the optimizers Mosek and Gurobi (using package DoE.MIParray's functions `gurobi_MIParray` and `mosek_MIParray`, respectively). In all cases, the correct GMA array was found. The run times are reported in Table 2. The table shows that running from resolution 1 or starting from the correct resolution does not make much difference for these small examples (although, in general, the difference can be substantial). It can also be seen that there is no general advantage of one software over the other. The run times we found are substantially shorter than those reported in Fontana (2017); a portion of the improvement is likely due to a more powerful computer, and small portions may be due to the fact that the latest version of Mosek is faster, but key contributors are most likely the smaller matrices and in particular the incorporation of the lower bound for confirming optimality. Fontana (2017) also reported a search time of 26 min for finding an OA(16, 2^6, 3); the time used for this optimization with our implementation was below 1 s, even starting with resolution 1 (but with exploiting the lower bound for confirming optimality), and regardless whether using Mosek or Gurobi.

Furthermore, Fontana (2017) created a GMA OA(18, $2^1 3^3$) (run time 270 s), a GMA OA(24, $2^2 3^1 4^1$, 2) (run time 37 min) and a GMA OA(12, $2^2 3^1 4^1$, 2) (run time 8 min). For the 18 run OA, the unique GMA array was found and confirmed within less than 1 s by both optimizers, even when starting from $R_0 = 1$ (using $k_{max} = 3$, which is sufficient for GMA in these designs with $m = 4$ factors). The array attains the lower bound $A_3 = 0.5$, which was already calculated in Section 2.3. For the 24 run array, our algorithm likewise produced a confirmed GMA OA(24, $2^2 3^1 4^1$, 2) in less than 1 s, since the array attains the lower bound $A_3 = 1/9$. For the 12 run array, our algorithm needed almost two minutes (Mosek) or about one minute (Gurobi) for obtaining the confirmed GMA array; the longer time is due to the fact that confirming GMA involved closing the gap for A_3 in this case, since there is no lower bound for A_3 in this resolution II design.

All examples in this section were quite small. We tried to run the Fontana algorithm on selected 72 run designs from the next section; this attempt was unsuccessful, as no solution was found in many hours.

Table 3: GMA A_3 values for the L18 series, their efficiencies w.r.t. the lower bound, proportion of GMA designs among all non-isomorphic designs, and creation times from Mosek

		OA(18, 3^a, 2)				OA(18, $2^1 3^a$, 2)			
a	LB	A_3	Eff. (%)	GMA/all	Time [s]	A_3	Eff. (%)	GMA/all	Time [s]
3	0.5	0.5	100.0	1/4	0.28	0.5	100.0	1/15	0.46
4	2.0	2.0	100.0	1/12	3.14	3.5	57.1	2/48	60.51
5	5.0	5.0	100.0	1/10	3.04	8.5	58.8	1/19	62.34
6	10.0	10.0	100.0	1/8	20.14	16.0	62.5	1/12	907.70
7	17.5	22.0	79.5	3/3	72.68	28.0	62.5	3/3	105.47

5.2 Further mixed level orthogonal arrays

Section 5.2.1 gives insights in the behavior of the lower bound relative to known GMA A_R values, and the algorithm's success in finding GMA A_R values in such cases. Furthermore, Section 5.2.2 provides a few optimal or close to optimal arrays for mixed level situations for which GMA A_R values are not known to us. The supplemental material exemplifies very easy and more difficult cases, inspired by an example from experimenting with Gurobi options.

5.2.1 Known series of arrays

The series OA(18, 3^a, 2) and OA(18, $2^1 3^a$, 2) are well-researched (see e.g. Schoen 2009), and all their GMA arrays are known. Table 3 shows the lower bounds for A_3 (which are the same for both series of arrays), the GMA A_3 values, the efficiencies of the lower bounds in comparison to these known optima, the number of non-isomorphic GMA arrays related to the overall number of non-isomorphic arrays, and the times [s] used by Mosek until either optimality of A_3 was confirmed or the time limit was reached (in most time limit cases, the optimal A_3 was actually found faster than the time limit). Optimization with Gurobi was also investigated (not shown), and was for most cases worse than with Mosek (exception: faster for $a = 7$). Clearly, the lower bound in Table 3 is much sharper for the symmetric designs. With Mosek, the algorithm finds the GMA designs in all cases; for OA(18, $2^1 3^6$, 2), a time limit of 900s was used (770s would have been sufficient), for all other cases the time limit was set to 60s. The time limit was applied per optimization problem, and times longer than the limit mean that problem (1) of Step 1 or its preparation in R package DoE.MIParray took up a relevant amount of time (noteworthy for $a = 7$). Confirmation within the time limit occurred for all cases of 100% efficiency, i.e. where the lower bound is sharp. For OA(18, $2^1 3^4$, 2), confirmation by closing the gap to 0 required about 464 min with Mosek; not shown in the table) or did not occur in more than 2525 min (gap still more than 50% with Gurobi, running with option `MIPfocus` set to 3 for improving the gap); for the other scenarii with less than 100% efficiency, confirmation has not been attempted. All arrays found are known from Schoen (2009) to have GMA. The table shows that the lower bound is far from perfect, particularly for mixed level cases, i.e. a failure to yield the lower bound does not imply a failure to yield the best design. Of course, for larger scenarii, optimizing the shortest word length does not generally imply that a GMA design will be found.

Table 4: Good 72 run designs of strength 2 found with our algorithm, using Mosek (m_j is the number of factors with j levels)

m_2	m_3	m_4	m_6	N/n	LB	A_3	Eff. (%)	Time [s]	order
1	1	1	1	2	0.111	0.111	100.0	3.6	
2	1	1	1	4	0.235	0.235	100.0	44.6	4,2,3,2,6
0	2	0	1	0.75	0.125	0.125	100.0	0.4	
1	2	0	1	1.5	0.125	0.125	100.0	12.1	
2	2	0	1	3	0.125	0.125	100.0	126.2	2,3,2,6,3
3	2	0	1	6	0.125	0.303	41.2	6736.0	2,6,3,3,2,2
4	2	0	1	12	0.125	0.473	26.4	46.6	2,3,6,3,2,2,2
2	2	1	0	2	0.012	0.012	100.0	3.6	
3	2	1	0	4	0.037	0.037	100.0	28.9	3,2,2,2,4,3
4	2	1	0	8	0.074	0.074	100.0	21.4	3,2,2,2,2,4,3
5	2	1	0	16	0.123	0.352	35.1	50.1	2,2,4,2,3,3,2,2
3	3	0	0	3	0.031	0.031	100.0	12.2	2,3,2,2,3,3
4	3	0	0	6	0.031	0.314	10.0	2210.0	
0	3	1	0	1.5	0.031	0.031	100.0	1.8	
1	3	1	0	3	0.031	0.031	100.0	47.6	
2	3	1	0	6	0.044	0.199	21.9	7.5	3,3,3,2,4,2
3	3	1	0	12	0.068	0.527	13.0	52.4	3,4,2,3,3,2,2
0	3	0	1	2.25	0.406	0.406	100.0	206.0	
1	3	0	1	4.5	0.406	0.469	86.7	7397.7	3,6,3,2,3
2	3	0	1	9	0.406	0.704	57.7	19.0	6,3,2,2,3,3
3	3	0	1	18	0.406	0.549	74.1	64.5	2,6,2,2,3,3,3

Some mixed level 48 run series with strength 3 or more, published by Eendebak and Schoen (2010), have been unsystematically investigated along similar lines as the 18 run series of Table 3. In all of them, the lower bound amounted to 100% of the GMA A_R for the smallest designs only, and the GMA A_R was found fast for most of the cases (regardless whether it was the lower bound or not), with cases being closer to saturation more difficult to find. In some cases, e.g. for OA(48, $2^8 3^1$, 3), the "looping through different level orders" strategy mentioned in Section 4.4 was needed for finding the best design (with each optimization step limited to 60s). This strategy will also be used for finding some new "good" designs in the next section.

5.2.2 Some new mixed level arrays

We initially focus on mixed level designs in $n = 72$ runs. $72 = 2^3 3^2$ offers many possibilities for accomodating mixed level designs, which are the target application of our algorithm. In this section, all optimizations used Mosek, which appeared to be successful fast in more situations than Gurobi.

In the light of the behavior regarding known GMA series, we expected to obtain the most useful new designs for situations close to the next higher strength, i.e. e.g. for strength 2 designs which are almost strength 3. Thus, we applied our algorithm for deriving 72 run arrays of strength 2 but close to strength 3 with confirmed GMA A_3 (equal to the lower bound) or low A_3 but unequal to the lower bound. These

Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

Table 5: Good large designs found with our algorithm, using Mosek (m_j is the number of factors with j levels)

n	m_2	m_3	m_4	m_6	N/n	R	LB	A_R	Eff. (%)	GMA	order	Search time [s]
144	1	2	2	0	2	3	0.012	0.012	100.0			
144	2	2	2	0	4	3	0.025	0.140	17.7			
216	1	2	1	1	2	3	0.012	0.064	19.4		3,3,6,4,2	60
288	0	2	2	1	3	3	0.008	0.177	4.4		4,4,3,6,3	120
432	1	3	2	0	2	3	0.001	0.018	7.7		3,4,3,3,2,4	360
192	3	1	2	0	2	5	0.111	0.111	100.0	*		
192	2	0	2	1	2	5	1	1.000	100.0	*		
192	1	1	3	0	2	4	0.111	0.111	100.0	*		
384	4	1	2	0	2	6	0.111	0.111	100.0	*	2,4,4,2,2,2,3	60
576	2	1	2	1	2	5	0.111	0.194	57.1		6,2,3,2,4,4	60
576	0	1	3	1	2	4	0.111	0.335	33.2		4,4,3,4,6	600

are shown in Table 4. Where the table shows an order, the default level order (from small to large) was not successful, and a search over level orders was applied, in most cases restricting the search time for each individual optimum to 60 s. In a few cases, for which the design obtained after 60 s was particularly unsatisfactory, the optimum of the search was subsequently optimized by a longer search (time limit up to three hours). Besides the quality of the resulting designs, the table also shows the search times needed for arriving at the observed optimum, when using the successful search order (of course, the total search takes much longer, if the successful order is not among the early ones). In one case, the search for a larger scenario (OA(72, $2^3 3^3 6^1$, 2)) yielded a lower A_3 value ($A_3 = 0.549$) than the search for a smaller one (OA(72, $2^2 3^3 6^1$, 2) with $A_3 = 0.704$). In this case, a column selection from the larger array of course yields a better array than the smaller one found with the algorithm alone. Two such column selections yield the best possible A_3; among these, one is preferrable because of a better A_4 value ($(A_3, A_4) = (0.493, 5.236)$). While some of the arrays that were found are quite good, others can be easily improved upon by the strategy of selecting columns from a published 72 run array with many columns, for example the OA(72, $2^4 3^3$, 2).

Besides the 72 run arrays, we considered a few larger cases, without systematically covering these. Table 5 lists scenarii with more than 100 runs, for which our algorithm ran successfully and found an array with confirmed optimal A_R or with "good" A_R in the sense that we were not able to find a better one elsewhere. For the only array with $A_R = 1$, a regular array with the same A_R can be obtained by the Kobilinsky et al. (2017) algorithm. The arrays with 100% efficiency have confirmed GMA if $m = m_2 + m_3 + m_4 + m_6 \leq R + 1$; this is due to the fact that optimization of A_{R+1} is obsolete because the sum of the GWLP is fixed to N/n. For the OA(144, $2^1 3^2 4^2$, 2), $m = 5 > R + 1$, and $A_{R+1} = A_4$ was optimized (in the sense of running the algorithm for eight hours, which substantially improved A_4 from the initial value 0.707 to 0.253), but not to a confirmed optimum; thus, the GMA status of that array is unknown, even though A_3 is a confirmed optimum. For those arrays for which A_R is not a confirmed optimum, optimization of A_{R+1} has not been attempted. The scenarii included in Table 5 were arbitrarily selected by considering large run sizes and a mix of numbers of levels with relatively small

size N of the corresponding full factorial, for which the Eendebak and Schoen website does not offer an array. (For example, an OA(192, $2^5 3^1 4^1$, 5) was not included, because it is available from the Eendebak and Schoen website.) The table does not report run times; for none of the arrays the construction took more than some hours (at most running overnight, e.g. for the optimization of A_3 *and* A_4 in the OA(144, $2^1 3^2 4^2$, 2), or for running 60 orders for up to 360s in each step for the OA(432, $2^1 3^3 4^2$, 2)), in most cases substantially less. Where Table 5 provides an order of levels, the array was found by a search over all different orders, allocating at most the time specified in the "Search time" column for each step of each order. The choice of a search time was to a certain extent arbitrary: where a short search time of 60s per run did not yield an acceptable result while improvements within some short time seemed likely, search time per run was increased to up to 10 minutes.

5.3 (Supersaturated) Resolution II arrays

Besides orthogonal arrays of strength at least 2, our algorithm can also produce (balanced) resolution II (strength 1) arrays that are not as widely used and published as orthogonal arrays. Several of the examples of Table 2 are of that nature. Balanced *supersaturated* arrays are special cases of such resolution II arrays, and Proposition 2 provided a lower bound for their A_2. Some sources provide supersaturated arrays (SSDs) with many more main effects df than there are experimental runs; we refrain from discussing the chances and risks of using SSDs and refer readers to Georgiou (2014) for such a discussion. Our algorithm is not suitable for the creation of arrays whose full factorials would be extremely large; smaller supersaturated arrays can, however, be successfully optimized with the help of our algorithm, since optimization of the sum or expectation of χ^2 (see also Section 2.3) is equivalent to optimization of A_2. Remember that $E(\chi^2) = nA_2/\binom{m}{2}$.

There are many smaller mixed level situations, for which a specific supersaturated array for the requirement at hand is not readily available. If a saturated OA can be found, a subset of its runs and columns can sometimes be used (see Xu 2015). An SSD for mixed level situations can also be constructed using the approach of augmenting an n run Hadamard matrix with an n-level factor; for example, the website by Gupta et al. (2011) lists an OA(12, $12^1 2^{11}$, 1) with $A_2 = 11$ and $E(\chi^2) = 2$. Such an array can be used for creating OA(12, $2^a 3^1 4^1$, 1); we used the `L12.2.11` from the Kuhfeld (2009) collection as imlemented in the R package DoE.base, augmented by a 12-level column. Table 6 shows optimized OA(12, $2^a 3^1 4^1$, 1), $a = 1, \ldots, 11$,

- from selecting 2-level columns from the L12 and replacing the 12 levels of the 12-level factor by a 3×4 full factorial (called the Hadamard matrix approach)
- in comparison to designs created with our algorithm, running it with time limit one hour (except for $a = 11$, for which the time limit was increased to five hours).

The designs with $a \leq 6$ have resolution II (strength 1), but are not supersaturated, the largest five designs are (slightly) supersaturated. The optimization details for the Hadamard approach are given in

Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

Table 6: OA(12, $2^a3^14^1$, 1) from Hadamard matrix or up to one hour (five hours) of our algorithm

		Lower bound		Hadamard approach				Our algorithm			
a	N	A_2	$E(\chi^2)$	Time [s]	$E(\chi^2)$	Eff. (%)	GR	Time [s]	$E(\chi^2)$	Eff. (%)	GR
1	24	0.111	0.444	10.9	0.444	100.0	2.67	0.2/ 3600	0.444	100.0	2.67
2	48	0.222	0.444	67.6	0.444	100.0	2.67	0.3/ 3600	0.444	100.0	2.67
3	96	0.333	0.400	242.6	0.400	100.0	2.67	0.3/ 3600	0.400	100.0	2.67
4	192	0.444	0.356	567.1	0.356	100.0	2.67	3.7/ 3600	0.356	100.0	2.67
5	384	0.556	0.317	901.4	0.413	76.9	2.59	2.0/ 3600	0.381	83.3	2.67
6	768	0.667	0.286	1015.9	0.500	57.1	2.59	164.6/ 3600	0.381	75.0	2.67
7	1536	0.778	0.259	803.0	0.537	48.3	2.59	31.3/ 3600	0.444	58.3	2.67
8	3072	1.188	0.317	436.6	0.593	53.4	2.29	97.1/ 3600	0.444	71.2	2.67
9	6144	1.910	0.417	160.2	0.655	63.7	2.18	438.5/ 3600	0.436	95.5	2.67
10	12288	2.729	0.496	35.7	0.727	68.2	2.00	3410.1/ 3600	0.566	87.7	2.59
11	24576	3.639	0.560	3.4	0.769	72.8	2.18	9324.2/18000	0.658	85.1	2.50

the supplementary material; note that the chosen route is a compromize between feasibility and coverage of all possibilities and does not guarantee that the best obtainable design is actually found; for the smallest four designs (known because of the lower bound) and the largest design (because all structurally different 3×4 full factorials were tried, run time almost six hours), a better design from the Hadamard approach does not exist, using full factorials for the 3×4 part of the array. Most designs created with our algorithm have a full factorial 3×4 part, but the largest two have not; thus, there might be a possibility to get a better design from the Hadamard approach by also trying mildly confounded 3- and 4-level factors, which would, however, dramatically increase the search space.

The MIP calculations were done with Mosek. The table shows the actual time needed for first reaching the optimum that was achieved within the time limit. Optimality was not confirmed for any of the arrays with $a \geq 5$. Our algorithm clearly outperformed the Hadamard approach:

- its $E(\chi^2)$ is closer to the lower bound for all cases with less than 100% efficiency,
- its generalized resolution (Grömping and Xu 2014) was at least as good, and much better for the cases with larger a (note that the GR for the Hadamard approach with a=10 can be improved to 2.184 by resolving ties in $E(\chi^2)$ by better GR; generally doing this increases search time).
- it was faster than the Hadamard approach for $a \leq 8$ (for $a = 1, 2, 3$, the Hadamard approach would have yielded the optimum design from a fast naïve optimization with just the lexicographic full factorial; for $a = 4$, the naïve approach was unsuccessful).

Since the bound on $E(\chi^2)$ is not necessarily sharp, a low value on efficiency does not necessarily mean a bad design, but may also arise from a poor lower bound. For $a = 7$, which has particularly low efficiency for both approaches, extending the search time of our algorithm to five hours did not improve the result; neither did a search over all 72 level orders (60 s each). The search found designs with $A_4 = 62/3$ or $A_4 = 188/9$; the default order yielded the slightly worse A_4 value. Trying to improve it with our algorithm (time limit one hour) was unsuccessful.

Accepted Manuscript for https://doi.org/10.1016/j.csda.2019.01.020 (CSDA 2019 Vol. 137 pp. 101-114)

6 Discussion

Mixed integer optimization can provide confirmed GMA arrays for small problems. For medium size problems, we have seen that it is often possible to obtain a confirmed optimum for the shortest word length A_R; this happens, whenever the GMA word length attains the lower bound provided in Proposition 1. It is then possible to further optimize the second shortest word length; however, for word lengths other than the shortest, there is generally no lower bound different from zero, so that it is usually necessary to confirm optimality by reducing the gap to zero, which is often infeasible, as was e.g. seen in the 72 run example of Section 3.

Obtaining optimized arrays by mixed integer optimization is particularly interesting for mixed level experimentation, in case published arrays do not cover the experimentation needs. Our algorithm is most useful, if the next larger strength is almost but not quite in reach. For such situations, a best available array is often not catalogued, and it is not unlikely that the lower bound of Proposition 1 is attained by the GMA A_R. Good arrays without optimality confirmation can also be obtained, e.g. for small supersaturated situations. Whenever optimality is not confirmed, it is advisable to compare the array obtained with our algorithm to arrays obtained by other approaches, e.g. by optimized column selection from a large nearly saturated array.

Conic quadratic mixed integer optimization is notoriously difficult. Besides playing with optimizer options (especially for Gurobi), the examples showed that the order in which the level numbers are provided to the optimizer can have a strong influence on success or failure of the optimization, and the R package DoE.MIParray provides a function that facilitates automated searches over level orders. This is a very pragmatic approach. It would be desirable to exploit the structure of the search space in some way. As the optimum order for the search was found to depend on aspects like the number of cores or the computational power of the machine used for the computations, existing general implementations of mixed integer optimization (like Mosek or Gurobi) can most likely not be easily adapted to exploit the structure of the search space. Thus, the task of developing an algorithm that exploits the structure of the search space for building a mixed level OA from scratch (without having to enumerate all interim design solutions) remains an open research question.

Our algorithm optimizes the summary measure A_R (or $E(\chi^2)$ for R=2), and does not consider worst case R factor sets. Nevertheless, in the resolution II 12 run designs considered in Section 5.3, it successfully produced arrays with good generalized resolution, i.e. with low worst case two factor confounding. It would be very useful if this behavior could be systematically enforced, rather than being an outcome that does or does not happen due to circumstances out of our control.

Acknowledgments

Ulrike Grömping's work was partially supported by Deutsche Forschungsgemeinschaft Grant GR 3843/2. We appreciate input by Hongquan Xu regarding validity of Proposition 3 for the sum of the A_i values in the mixed-level case.

7 References

Ai, M., Fang, K.-T. and He, S. (2007). $E(\chi^2)$-optimal mixed-level supersaturated designs. *Journal of Statistical Planning and Inference* **137**, 306-316.

Bailey, R.A.B. (1982). The decomposition of treatment degrees of freedom in quantitative factorial experiments. *Journal of the Royal Statistical Society* **B 44**, 63-70.

Bulutoglu, D.A. and Margot, F. (2008). Classification of orthogonal arrays by integer programming. *Journal of Statistical Planning and Inference* **138**, 654-666.

Butler, N.A. (2005). Generalised minimum aberration construction results for symmetrical orthogonal arrays. *Biometrika* **92**, 485-491.

Cheng, S.-W. and Ye, K.Q. (2004). Geometric isomorphism and minimum aberration for factorial designs with quantitative factors. *The Annals of Statistics* **32**, 2168-2185.

Eendebak, P. T. and Schoen, E. D., (2010). Orthogonal Arrays Website.
http://www.pietereendebak.nl/oapackage/series.html. Last accessed February 27, 2017.

Fontana, R. (2013). Algebraic generation of minimum size orthogonal fractional factorial designs: an approach based on integer linear programming. *Computational Statistics* **28**, 241-253.

Fontana, R. (2017). Generalized minimum aberration mixed-level orthogonal arrays: A general approach based on sequential integer quadratically constrained quadratic programming. *Communications in Statistics –Theory and Methods* **46**, 4275-4284.

Georgiou, S.D. (2014). Supersaturated designs: a review of their construction and analysis. *Journal of Statistical Planning and Inference* **144**, 92 - 109.

Grömping, U. (2017). Frequency tables for the coding invariant quality assessment of factorial designs. *IISE Transactions* **49**, 505-517.

Grömping, U. (2018a). Coding Invariance in Factorial Linear Models and a New Tool for Assessing Combinatorial Equivalence of Factorial Designs. *Journal of Statistical Planning and Inference* **193**, 1-14.

Grömping, U. (2018b). DoE.MIParray: Creation of Arrays by mixed integer optimization. R package version 0.10. In: R Core Team (2018).

Grömping, U. (2018c). R Package **DoE.base** for Factorial Designs. *Journal of Statistical Software* **85**(5), 1-41.

Grömping, U. and Xu, H. (2014). Generalized resolution for orthogonal arrays. *The Annals of Statistics* **42**, 918-939.

Gupta, V.K., Parsad, R., Kole, B. and Bhar, L.M. (2011). Supersaturated Designs. http://www.iasri.res.in/design/Supersaturated_Design/SSD/Supersaturated.html. Indian Agricultural Statistics Research Institute (ICAR), New Delhi 110 012, India. Last accessed September 22, 2017.

Gurobi Optimization, Inc. (2017). Gurobi Optimizer Reference Manual. http://www.gurobi.com/documentation/current/refman.pdf.

Hedayat, S., Sloane, N.J. and Stufken, J. (1999). *Orthogonal Arrays: Theory and Applications*. Springer, New York.

Kobilinsky, A., Monod, H. and Bailey, R.A. (2017). Automatic generation of generalised regular factorial designs. *Computational Statistics and Data Analysis* **113**, 311-329.

Kobilinsky, A., Bouvier, A. and Monod, H. (2018). planor: an R package for the automatic generation of regular fractional factorial designs. R package version 1.3-9. In: R Core Team (2018).

Kuhfeld, W. (2009). Orthogonal Arrays. Website courtesy of SAS Institute http://support.sas.com/techsup/technote/ts723.html. Last accessed September 25 2017.

Liu, M.Q. and Lin, D.K.J. (2009). Construction of Optimal Mixed-Level Supersaturated Designs. *Statistica Sinica* **19**, 197-211.

Mosek ApS (2017). MOSEK Rmosek Package 8.1.0.24. http://docs.mosek.com/8.1/rmosek/index.html.

Mosek ApS (2018). MOSEK Modeling Cookbook - Release 3.0. https://docs.mosek.com/modeling-cookbook/index.html (accessed October 12, 2018).

Nguyen, M.V.M. (2008). Some new constructions of strength 3 mixed orthogonal arrays. *Journal of Statistical Planning and Inference* **138**, 220–233.

NIST/SEMATECH (2016). *e-Handbook of Statistical Methods*, http://www.itl.nist.gov/div898/ handbook/, 21 August 2016.

R Core Team (2017). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.

Schoen, E.D. (2009). All orthogonal arrays with 18 runs. *Quality and Reliability Engineering International*

25, 467-480.

Schoen, E. D., Eendebak, P. T. and Nguyen, M. V. M. (2010). Complete enumeration of pure-level and mixed-level orthogonal arrays. *Journal of Combinatorial Designs* **18**, 123–140. doi:10.1002/jcd.20236.

Vasilev, N., Schmitz, Ch., Grömping, U., Fischer, R. and Schillberg, S. (2014). Assessment of Cultivation Factors that Affect Biomass and Geraniol Production in Transgenic Tobacco Cell Suspension Cultures. *PLoS ONE* **9**(8): e104620. DOI:10.1371/journal.pone.0104620.

Xu, H., Cheng, S.-W. and Wu, C.F.J. (2004). Optimal projective three-level designs for factor screening and interaction detection. *Technometrics* **46**, 280–292.

Xu, H. (2005). A Catalogue of Three-Level Regular Fractional Factorial Designs. *Metrika* **62**, 259-281.

Xu, H. (2015). Nonregular factorial and supersaturated designs. In *Handbook of Design and Analysis of Experiments*, eds A. Dean, M. Morris, J. Stufken and D. Bingham, Chapman and Hall/CRC, Series: Chapman & Hall/CRC Handbooks of Modern Statistical Methods, 339-370.

Xu, H. and Wu, C.F.J. (2001). Generalized minimum aberration for asymmetrical fractional factorial designs [corrected republication of MR1863969]. *The Annals of Statistics* **29**, 1066–1077.