A canonical form for non-regular arrays based on generalized wordlength pattern values of delete-one-factor projections

P.T. Eendebak*

April, 2014

Abstract

We introduce a canonical representative for the isomorphism classes of non-regular orthogonal arrays based on the generalized wordlength patterns (GWLP) of delete-one-factor projections. These GWLP values have been used recently to introduce a fast isomorphism test for two-level regular arrays. We show that the delete-one-factor projection GWLP method can be adopted to reduce both regular and non-regular orthogonal arrays to canonical form.

The new canonical form is used in an existing framework to generate minimal complete sets of non-symmetric non-regular arrays. We show that the new method is efficient for reduction to canonical form, but not suitable for generating minimal complete sets.

KEY WORDS: Experimental Design; Orthogonal Array; Delete-one-factor; Canonical form; Generalized Wordlength Pattern

*E-mail: pieter.eendebak@gmail.com. Address: University of Antwerp, Departement Engineering Management, Prinsstraat 13, 2000 Antwerp, Belgium.

1 Introduction

Orthogonal arrays are an important tool in the design of experiments [6]. Formally, an orthogonal array of strength t is an $N \times k$ matrix whose jth column contains s_j different factor-levels in such a way that, for any t columns, every t-tuple of levels appear equally often in the matrix [9]. Two arrays are said to be isomorphic if one array can be obtained by permuting rows, columns, and/or factor levels of the other array.

A major step in the analysis of orthogonal arrays is to generate representatives for all isomorphism classes of a specific type of orthogonal arrays. The type is usually specified by the number of runs, the number of factors, the factor levels and the strength. A *minimal complete set* (MCS) for a specific type of arrays is a set of arrays with exactly one representative for each isomorphism class.

A generic method for generating these isomorphism classes is to start with a minimal complete set of arrays with a specified number of columns, say k, and then extend this set to a complete set of arrays with $k + 1$ columns. In this method two main components can be identified:

- **Extension** A method is specified to generate arrays with $k + 1$ columns from the set of arrays with k columns. This method must guarantee that for each isomorphism class for $k+1$ columns at least 1 array is generated.
- **Reduction** From the set of generated arrays a minimum complete set has to be generated. This can be done by either transforming the arrays to a canonical form or comparing the arrays pairwise and performing an isomorphism check.

The most basic isomorphism check between two arrays is to test all possible transformations of one of the arrays (exhaustive isomorphism testing). This includes row, column and level permutations. An isomorphism check using this method is computationally very expensive.

We present a short overview of other works using the extension and reduction method. The extension method was used by Chen et al. [2] for symmetrical regular designs. Their extension method and reduction method use properties specific to regular designs. Sun et al. [12] present an algorithm to construct all non-isomorphic two-level designs of specified run-size and numbers of factors. Like Chen et al. [2], the authors start with a minimum complete set of designs with a certain number of factors and they consider all possible extensions with one additional column. The resulting designs are then classified with the extended word-length pattern [3]. Those belonging to the same class are further tested with algebraic techniques.

By defining canonical forms for the isomorphism classes the reduction step can be performed more efficiently. Canonical forms for non-symmetric arrays are presented by Bulutoglu and Margot [1] and Schoen et al. [11]. The canonical form in [1] is based on a canonical form for graphs based on Nauty [7, 8] while the canonical form in [11] is based on the lexicographic ordering of arrays. Both approaches lead to the same representatives for the isomorphism classes.

Recently, a series of papers has appeared [13, 10] which use the GWLP values of delete-one-factor projections as an alternative ordering of the columns. Using this method impressive results have been obtained for regular fractional factorial arrays. The new ordering can be extended to non-symmetric as well as non-regular arrays.

In Section 2 we introduce generalized wordlength patterns, delete-one-factor projections and the new canonical form for orthogonal arrays. Using this canonical form a new method to generate minimal complete sets is presented based on the generic framework described above. In Section 3 we compare the efficiency of the new method to the method of Schoen et al. [11].

2 Method

In this section we introduce delete-one-factor projections and generalized wordlength patterns and we define the canonical form for non-regular arrays. We modify the framework in [11] to generate all isomorphism classes of non-symmetric non-regular orthogonal arrays.

2.1 Preliminaries

To describe the modified algorithm, we need some definitions. We first introduce orthogonal arrays. Next, we introduce GWLPs (the theory here is from Xu and Wu [14]) and the ordering of arrays leading to canonical forms.

Definition 1. A symmetric orthogonal array (OA) of strength t, N runs and k factors at s levels is an $N \times k$ array of $0, \ldots, (s-1)$-valued symbols such that for every t columns every t-tuple occurs equally often [9]. The set of all OAs with given strength, runs and levels is denoted by $\mathrm{OA}(N; s^k; t)$.

Two arrays are said to be combinatorially isomorphic if one array can be obtained by permuting rows, columns, or factor levels of the other array.

An $N \times k$ design D consists of a set of row vectors of length k. For two row vectors in D, say a and b, we denote by $d_H(a, b)$ for the Hamming distance between a and b. We define the binomial coefficients as $\binom{n}{k} = n!/((n-k)!k!)$. We use the convention that $0! = 1$.

Definition 2 (Distance distribution)**.** Let D be an $N \times k$ matrix. For $j = 0, \ldots, k$ we define

$$B_j(D) = N^{-1}|\{(a, b) : d_H(a, b) = j, a \in D, b \in D\}|.$$

The *distance distribution* of D is defined as $(B_0(D), \ldots, B_k(D))$. The *MacWilliams*

transforms of the distance distribution are defined as

$$B'_j(D) = N^{-1} \sum_{i=0}^{k} B_i(D) P_j(i; k, s)$$

where $P_j(x; k, s) = \sum_{i=0}^{j} (-1)^i (s-1)^{j-i} \binom{x}{i} \binom{k-x}{j-i}$ are the Krawtchouk polynomials.

Definition 3 (Generalized Wordlength Pattern)**.** For an (N, s^k)-design D the generalized wordlength pattern of D is equal to $A(D) = (B'_0(D), B'_1(D), \ldots, B'_k(D))$.

For regular 2-level arrays the value $A_i(D)$ is equal to the number of words of length i in the defining contrast subgroup of D [13]. For any 2-level array our definition of the generalized wordlength pattern is equivalent to

$$A_i(D) = N^{-2} \sum_{\mathrm{wt}(u)=i} |J_u(D)|^2 \tag{1}$$

with the J-characteristic $J_u(D) = \sum_{x \in D} (-1)^{\langle u, x \rangle}$. The summation in Equation 1 is over all k-tuple binary vectors u with i nonzero elements.

Definition 4 (GWLP ordering)**.** Let $a = (a_0, \ldots, a_k)$ and $b = (b_0, \ldots, b_k)$ be two generalized wordlength patterns. We order the GWLPs by the lexicographic ordering. So $a < b$ if there is an l such that $a_j = b_j$ for $j = 1, \ldots, l-1$ and $a_l < b_l$.

In statistical applications, designs with a small GWLP are desirable. The arrays with strength t have $a_0 = 1$ and $a_1 = \ldots = a_t = 0$.

Let X be an $N \times k$ array. For $1 \leq j \leq k$ we define $\pi_j(X)$ to be the array obtained by deleting the jth column from X. With $d_j(X)$ we denote the GWLP of $\pi_j(X)$.

2.2 Canonical form

We introduce several orderings of the set of orthogonal arrays. For each isomorphism class of $\mathrm{OA}(N; s; t)$ an ordering defines a unique minimal element. The

minimal element for each isomorphism class defines a canonical form for that particular isomorphism class.

Definition 5 (LMC ordering)**.** Let X and Y be two $N \times k$ arrays. Let x be the $N \cdot k$ tuple obtained by concatenating the columns of X, let y be the tuple obtained by concatenating the columns of Y. We define X to be smaller than Y in the LMC ordering (lexicographically minimum in columns ordering) if there is an $l \leq Nk$ such that $x_i = y_i$ for $i < l$ and $x_l < y_l$.

The LMC ordering was used in [11].

Definition 6 (Delete-one-factor ordering)**.** Let X and Y be two $N \times k$ arrays. We define X to be smaller than Y in the *delete-one-factor ordering* if there is an l such that $d_j(X) = d_j(Y)$ for $j = 1, \ldots, l-1$ and $d_l(X) < d_l(Y)$ or $d_j(X) = d_j(Y)$ for $j = 1, \ldots, k$ and X is lexicographically smaller than Y.

The delete-one-factor ordering is based on the GWLPs of the delete-one-factor projections and on the LMC ordering. This new ordering defines for each isomorphism class a unique element which is minimal according to this ordering.

Example 1 (Delete-one-factor values). Consider the following array in lexico-

graphically minimal form in OA$(12; 2^7; 2)$

$$X = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The delete-one-factor generalized wordlength patterns are given by:

Deleted column	GWLP
1	$d_1(X) = (1, 0, 0, 2.22, 1.67, 0.444, 0)$
2	$d_2(X) = (1, 0, 0, 2.22, 1.67, 0.444, 0)$
3	$d_3(X) = (1, 0, 0, 2.22, 1.67, 0.444, 0)$
4	$d_4(X) = (1, 0, 0, 2.22, 1.67, 0, 0.444)$
5	$d_5(X) = (1, 0, 0, 2.22, 1.67, 0.444, 0)$
6	$d_6(X) = (1, 0, 0, 2.22, 1.67, 0.444, 0)$
7	$d_7(X) = (1, 0, 0, 2.22, 1.67, 0.444, 0)$

Since the delete-one-factor values are not ordered, this array is not in minimal according to the delete-one-factor ordering. ▲

Using the delete-one-factor ordering, we can create a powerful algorithm to reduce an array from OA$(N; s^k; t)$ to minimal form. This algorithm basically consists of 2 steps: sorting the columns using the delete-one-factor GWLPs and then reducing the array further using LMC ordering. The algorithm is summarized in Algorithm 1.

Delete-one-factor reduction

Input: Orthogonal array X
Output: Delete-one-factor canonical form of the array

1. For each column k, calculate the delete-one-factor GWLP $d_k(X)$. Sort the columns of the array in increasing order of the delete-one-factor GWLPs.
2. Reduce the resulting array to lexicographically minimal form while respecting the ordering imposed by the delete-one-factor GWLPs.

Algorithm 1: Delete-one-factor reduction

The first step can be performed by calculating the delete-one-factor GWLPs and sorting these using the ordering on GWLPs. Next, we explain how to perform the reduction while respecting the ordering. Let X be an array in $\mathrm{OA}(N; s^k; t)$ and assume the columns have been ordered with the delete-one-factor ordering. The delete-one-factor GWLPs can be written as $d_1(X), \ldots, d_k(X)$ where $d_1(X) \leq d_2(X) \leq \ldots \leq d_k(X)$. Let $b = (b_1, \ldots, b_n)$ be the vector such that $d_1(X) = d_2(X) = \ldots = d_{b_1}(X)$, $d_{b_1+1}(X) = d_{b_1+2}(X) = \ldots = d_{b_1+b_2}(X)$, etc. We can then consider the array in $\mathrm{OA}(N; s^k; t)$ to be an element from $\mathrm{OA}(N; s^{b_1} s^{b_2} \cdots s^{b_n}; t)$. We reduce this array to lexicographically minimal form using the method of [11]. The software used contains functions [5, see the `arraydata_t` structure] to impose the ordering of the columns. These functions set the column structure of the array equal to the structure of the symmetry group obtained from the delete-one-factor projections.

The method described above has a computational advantage over the original LMC method. Since the columns with different delete-one-factor values cannot be interchanged any more, the number of column permutations that has to be checked in order to perform the reduction to minimal form is greatly reduced. This is illustrated in Example 3 on page 13.

Definition 7. Let ρ be a method for reducing an array to some canonical form and let $\pi_{-1}()$ be the projection of an array onto the first $k-1$ factors (by deletion of the last column). We say that the canonical form ρ is *stable under*

column extensions if for any extension Y of X we have

$$\rho(X) = \pi_{-1}(\rho(Y)). \tag{2}$$

The LMC canonical form of an array is stable under column extensions. The delete-one-factor form is not stable, as illustrated in Example 2.

Example 2 (Delete-one-factor canonical form). Let the strength 2 orthogonal array Y be defined by

$$Y = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0
\end{pmatrix}$$

and let $X = \pi_{-1}(Y)$ be the matrix obtained by deleting the last column from Y. The array Y is in canonical form for the delete-one-factor ordering. The

delete-one-factor GWLP values are

$$d_1(Y) = (1.00, 0.00, 0.00, 1.50, 1.00, 0.50, 0.00),$$
$$d_2(Y) = (1.00, 0.00, 0.00, 1.50, 1.00, 0.50, 0.00),$$
$$d_3(Y) = (1.00, 0.00, 0.00, 1.50, 1.00, 0.50, 0.00),$$
$$d_4(Y) = (1.00, 0.00, 0.00, 1.75, 0.75, 0.25, 0.25),$$
$$d_5(Y) = (1.00, 0.00, 0.00, 1.75, 0.75, 0.25, 0.25),$$
$$d_6(Y) = (1.00, 0.00, 0.00, 2.00, 1.00, 0.00, 0.00),$$
$$d_7(Y) = (1.00, 0.00, 0.00, 2.00, 2.00, 0.00, 0.00).$$

Hence, the GWLP values are ordered. However,

$$d_1(X) = (1.00, 0.00, 0.00, 1.00, 0.50, 0.00),$$
$$d_2(X) = (1.00, 0.00, 0.00, 1.00, 1.00, 0.00),$$
$$d_3(X) = (1.00, 0.00, 0.00, 1.00, 1.00, 0.00),$$
$$d_4(X) = (1.00, 0.00, 0.00, 1.00, 0.50, 0.00),$$
$$d_5(X) = (1.00, 0.00, 0.00, 1.00, 0.50, 0.00),$$
$$d_6(X) = (1.00, 0.00, 0.00, 1.00, 0.50, 0.00).$$

As a result, the GWLP values for X are not ordered, so X is not in canonical form for the delete-one-factor ordering. We conclude that the delete-one-factor ordering is not stable under column extensions. ▲

2.3 Minimum complete set algorithm

In this section, we present two algorithms to generate a minimum complete set for OA$(N; s^k; t)$. The first algorithm is given in Algorithm 2. This algorithm is based on the LMC normal form and was introduced in [11]. In practice, it is not necessary to generate all the extensions in step 2a of Algorithm 2. The

requirement for the algorithm to work is that the generated set of extensions contains, for each isomorphism class in $\mathrm{OA}(N; s^{j+1}; t)$, the canonical form. We can use the methods described in [11] to reduce the number of extensions. For example, when extending an array with k columns we can let the algorithm generate only extension columns which are lexicographically larger than column the last column of the array. All extension columns which are lexicographically smaller than the last column can be discarded since these columns will not produce arrays in canonical form.

LMC Minimal Complete Set algorithm

Input: Specification of N, t, s and k.
Output: Minimal complete set for $\mathrm{OA}(N; s^k; t)$.

1. Start with the root array of $\mathrm{OA}(N; s^t; t)$. The root forms a MCS for $\mathrm{OA}(N; s^t; t)$.
2. For $j = t, \ldots, k-1$ extend the MCS for $\mathrm{OA}(N; s^j; t)$ to a MCS for $\mathrm{OA}(N; s^{j+1}; t)$ using the following procedure:
   (a) For each array, create all possible extensions to $j+1$ columns.
   (b) For all extensions, check whether the array is in LMC form
   (c) Discard the arrays not in LMC form. The remaining arrays form a MCS for $\mathrm{OA}(N; s^{j+1}; t)$.

Algorithm 2: The LMC MCS algorithm

A second algorithm to generate a MCS is specified in Algorithm 3. This algorithm generates the MCS with representatives in the delete-one-factor canonical form.

For this algorithm we can use some of the methods described in [11] to reduce the number of arrays that has to be generated. The requirement is that, for each isomorphism class, there is at least one representative array within the set of generated arrays. A method we cannot use to reduce the generation of arrays is to discard the extension columns which are lexicographically smaller then the previous column. The reason for this is that the delete-one-factor normal form is not stable under columns extensions.

A crucial difference between Algorithm 2 and Algorithm 3 is that, in Algo-

Delete-one-factor Minimal Complete Set algorithm

Input: Specification of N, t, s and k.
Output: Minimal complete set for $\mathrm{OA}(N; s^k; t)$.

1. Start with the root array of $\mathrm{OA}(N; s^t; t)$. The root forms a MCS for $\mathrm{OA}(N; s^t; t)$.
2. For $j = t, \ldots, k-1$ extend the MCS for $\mathrm{OA}(N; s^j; t)$ to a MCS for $\mathrm{OA}(N; s^{j+1}; t)$ using the following procedure:
   (a) For each array, create all possible extensions to $j+1$ columns.
   (b) For all extensions, reduce the array to delete-one-factor normal form using Algorithm 1.
   (c) From the resulting set of arrays only keep the unique elements. The set of unique arrays forms a MCS for $\mathrm{OA}(N; s^{j+1}; t)$.

Algorithm 3: The delete-one-factor MCS algorithm

rithm 2, we can check whether the generated arrays are in canonical form, but, in Algorithm 3, we have to reduce the arrays to canonical form. The reason we can use the check in Algorithm 2 is that the LMC canonical form is stable under column extensions.

The algorithm can be extended to non-symmetric arrays. For non-symmetric arrays, we can use the definitions from [14] to define the GWLPs. We refer to Appendix A for details.

3 Results

The methods described in the previous section have been implemented in C++ with a command line as well as a Python interface. The code for analyzing arrays is contained in the Orthogonal Array package [5] and is available online [4]. Example code to analyse some of the arrays is included in Appendix B. We compare two applications of both methods:

- Reduction of an array to normal form.
- Generation of a MCS for a certain class of arrays.

3.1 Reduction to canonical form

The cases for which the delete-one-factor method works well, are the cases for which the delete-one-factor GWLPs have a large variation. If the GWLPs are all equal, then the GWLPs impose no additional constraints on the ordering of the columns of the array and the ordering reduces to the LMC ordering.

We consider the cases $\mathrm{OA}(32; 2^a; 3)$ and $\mathrm{OA}(40; 2^a; 3)$. First we consider the reduction of arrays in this class to canonical form. We do this by taking a representative for each isomorphism class in $\mathrm{OA}(N; 2^a; 3)$, for various a, applying a random transformation of the rows, columns, and levels to this representative and then measuring the time needed for reducing the array to canonical form. The procedure is performed several times to eliminate the effect of the random component. The results for reduction of a randomized array to canonical form are given in Table 1 and Table 2.

From the tables it is clear that the delete-one-factor method performs better in all cases. The improvement is relatively larger number for a higher number columns. The reason is that for a larger number of columns the delete-one-factor projection values have enough variation to reduce the number of column permutations. In Table 3, the structure of the delete-one-factor GWLP groups is shown for $\mathrm{OA}(32; 2^9; 3)$. The symmetry groups have been calculated with the Orthogonal Array package (see Appendix B for an example calculation). It is clear that for most arrays there is enough structure in the group to reduce the number of column permutations that have to be analysed. The average size of the column permutation groups is 63611, which is about 18% of the full column permutation group size which is $9! = 362880$.

Example 3 (Column permutations in normal form reduction). Let X be the array in $\mathrm{OA}(32; 2^9; 3)$ defined by

$$X = \begin{pmatrix} 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1 \\ 0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1 \\ 0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1 \\ 0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1 \\ 0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1 \\ 0\,0\,1\,1\,0\,0\,1\,1\,1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,0\,0\,1\,1\,0\,0\,1\,1 \\ 0\,0\,1\,1\,1\,1\,0\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,1\,1\,0\,0\,0\,0\,1\,1 \\ 0\,1\,0\,1\,0\,1\,0\,0\,1\,1\,1\,1\,0\,0\,1\,1\,0\,0\,0\,0\,1\,1\,0\,1\,0\,1\,0\,1 \\ 0\,1\,0\,1\,1\,0\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\,1\,0\,0\,1\,0\,1 \end{pmatrix}^T.$$

The GWLPs of the delete-one-factor projections are:

Deleted column	GWLP
1	$d_1(X) = (1.0, 0.0, 0.0, 0.0, 5.0, 0.0, 2.0, 0.0, 0.0)$
2	$d_2(X) = (1.0, 0.0, 0.0, 0.0, 5.0, 0.0, 2.0, 0.0, 0.0)$
3	$d_3(X) = (1.0, 0.0, 0.0, 0.0, 5.25, 0.0, 1.5, 0.0, 0.25)$
4	$d_4(X) = (1.0, 0.0, 0.0, 0.0, 5.25, 0.0, 1.5, 0.0, 0.25)$
5	$d_5(X) = (1.0, 0.0, 0.0, 0.0, 5.25, 0.0, 1.5, 0.0, 0.25)$
6	$d_6(X) = (1.0, 0.0, 0.0, 0.0, 5.25, 0.0, 1.5, 0.0, 0.25)$
7	$d_7(X) = (1.0, 0.0, 0.0, 0.0, 6.0, 0.0, 1.0, 0.0, 0.0)$
8	$d_8(X) = (1.0, 0.0, 0.0, 0.0, 6.0, 0.0, 1.0, 0.0, 0.0)$
9	$d_9(X) = (1.0, 0.0, 0.0, 0.0, 7.0, 0.0, 0.0, 0.0, 0.0)$

The structure of the column permutation group is [2, 4, 2, 1].

The array is from $\mathrm{OA}(32; 2^9; 3)$, but after calculation of the delete-one-factor projection values we can consider the array to be an element of $\mathrm{OA}(32; 2^2 2^4 2^2 2^1; 3)$. With a naive algorithm for reduction to canonical form, the number of column permutations that has be checked is $9! = 362880$. With the LMC algorithm from [11] the columns are selected one at a time and the tree is pruned as soon as possible. For the first 3 columns there is no pruning, since there is no structure because of the strength of 3. Most column permutations can be discarded at the fourth of fifth column. A rough estimate of the number of column permutation to be checked is $5!\binom{9}{5} = 15120$. If we consider the delete-one-factor projection values, the number of column permutations to be checked is reduced

Case	Number of classes	Reduction time LMC	Reduction time DOP
$\mathrm{OA}(32, 3, 2^6)$	10	6.2 [ms]	4.3 [ms]
$\mathrm{OA}(32, 3, 2^7)$	17	6.5 [ms]	3.6 [ms]
$\mathrm{OA}(32, 3, 2^8)$	33	11.5 [ms]	8.3 [ms]
$\mathrm{OA}(32, 3, 2^9)$	34	17.3 [ms]	11.4 [ms]
$\mathrm{OA}(32, 3, 2^{10})$	32	43.6 [ms]	26.0 [ms]
$\mathrm{OA}(32, 3, 2^{11})$	22	86.9 [ms]	28.9 [ms]
$\mathrm{OA}(32, 3, 2^{12})$	23	210.5 [ms]	166.4 [ms]
$\mathrm{OA}(32, 3, 2^{13})$	12	444.5 [ms]	347.2 [ms]

Table 1: Mean calculation times for reducing randomized arrays to normal form.

Case	Number of classes	Reduction time LMC	Reduction time DOP
$\mathrm{OA}(40, 3, 2^7)$	25	4.3 [ms]	1.5 [ms]
$\mathrm{OA}(40, 3, 2^8)$	105	5.5 [ms]	2.3 [ms]
$\mathrm{OA}(40, 3, 2^9)$	213	7.6 [ms]	2.0 [ms]
$\mathrm{OA}(40, 3, 2^{10})$	353	12.9 [ms]	3.4 [ms]
$\mathrm{OA}(40, 3, 2^{11})$	260	22.8 [ms]	1.6 [ms]
$\mathrm{OA}(40, 3, 2^{12})$	235	40.6 [ms]	9.1 [ms]
$\mathrm{OA}(40, 3, 2^{13})$	132	72.3 [ms]	8.3 [ms]
$\mathrm{OA}(40, 3, 2^{14})$	96	153.9 [ms]	13.1 [ms]
$\mathrm{OA}(40, 3, 2^{15})$	36	324.9 [ms]	23.7 [ms]

Table 2: Mean calculation times for reducing randomized arrays to normal form.

even further. The first two columns are the columns with lowest GWLP value. There are precisely 2! possible permutations for these first 2 columns. For the next 4 columns we have 4! combinations, etc. In total we have 2!4!2!1! = 96 column permutations that have to be checked. ▲

The running time of the algorithm is to a large extent determined by the number of column permutations that has to be checked. For the class $\mathrm{OA}(40; 2^a; 3)$ a rough estimate of the number of column permutations to be checked with the LMC ordering is $5!\binom{n}{5}$. A graph of this complexity together with the computation times per array for the LMC and the delete-one-factor method is given in Figure 1. The complexity estimate is scaled such that the mean complexity and mean computation time of the LMC method are equal. We can see that the computing times for the LMC method scale with the number of columns roughly as the complexity estimate.

Array index	Group structure	Group size
0	[8, 1]	40320
1	[6, 1, 2]	1440
2	[3, 4, 2]	288
3	[7, 2]	10080
4	[6, 2, 1]	1440
5	[8, 1]	40320
6	[5, 1, 2, 1]	240
7	[2, 4, 2, 1]	96
8	[6, 2, 1]	1440
9	[4, 1, 4]	576
10	[5, 2, 2]	480
11	[5, 2, 2]	480
12	[6, 2, 1]	1440
13	[8, 1]	40320
14	[3, 2, 4]	288
15	[9]	362880
16	[9]	362880
17	[9]	362880
18	[4, 3, 1, 1]	144
19	[8, 1]	40320
20	[4, 1, 4]	576
21	[4, 2, 2, 1]	96
22	[4, 2, 2, 1]	96
23	[3, 4, 1, 1]	144
24	[4, 4, 1]	576
25	[4, 4, 1]	576
26	[8, 1]	40320
27	[3, 2, 4]	288
28	[9]	362880
29	[9]	362880
30	[7, 1, 1]	5040
31	[8, 1]	40320
32	[8, 1]	40320
33	[8, 1]	40320

Table 3: Structure of delete-one-factor symmetry group for the arrays in $\mathrm{OA}(32, 3, 2^9)$. The mean column permutation group size is 63611.3.

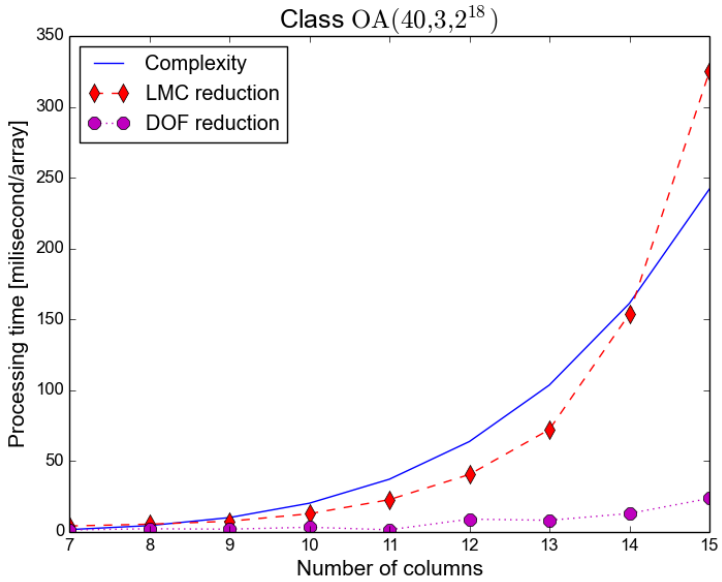

Figure 1: Mean reduction time per array.

3.2 Extension results

Recall that the methods to determine all isomorphism classes in $\mathrm{OA}(N; 2^a; t)$ are based on two steps:

- **Extension** Generate a set of arrays that includes at least one representative for each isomorphism class.
- **Reduction** Reduce the set of arrays so that each class has only one representative.

In this section we compare two such methods. The first method (Algorithm 2) is the method from [11]), the second method (Algorithm 3) is similar to the first, but uses the new canonical form described in Section 2.2. The running times of both algorithms depend on both the extension and reduction part. The ratio between these part can differ between different classes of orthogonal arrays.

For the delete-one-factor method we have implemented one additional step to increase the computation speed. For each extension Y of an array X generated,

the delete-one-factor GWLPs $d_j(Y)$ are calculated, and, for an array in delete-one-factor canonical form these values are decreasing. If we find an extension for which the final column k satisfies $d_k(Y) > d_{k-1}(Y)$, then we can discard this array. The reason is that the array will be generated also as an extension of the canonical form of the array $\pi_j(X)$ as well. The canonical form of the array $\pi_j(X)$ is not equal to Y (since they have different GWLPs). Since we only need to generate at least one representative for each isomorphism class, discarding the array Y does not influence the MCS generated.

The results for extension of the classes $\mathrm{OA}(32; 2^9; 3)$ and $\mathrm{OA}(40; 2^9; 3)$ with one additional factor are given in Table 4 and Table 3.2. The delete-one-factor method generates more arrays to be checked. This results in a higher computation time.

	LMC method	DOP method
# input arrays	34	34
# generated extensions	217	292
# reduced extensions	32	32
Processing time	0.9 [s]	5.1 [s]

Table 4: Results for extension of $\mathrm{OA}(32; 2^9; 3)$ to $\mathrm{OA}(32; 2^{10}; 3)$.

	LMC method	DOP method
# input arrays	213	213
# generated extensions	1844	3105
# reduced extensions	353	353
Processing time	3.5 [s]	4.8 [s]

Table 5: Results for extension of $\mathrm{OA}(40; 2^9; 3)$ to $\mathrm{OA}(40; 2^{10}; 3)$.

4 Discussion

Experiments show that with the new delete-one-factor ordering a reduction to canonical form can be performed much faster than with the lexicographic ordering. However, this new ordering does not work well with the current state-of-the-art extension algorithms. In the extension phase much more arrays are

generated, so the overall calculation is not faster than the original lexicographic ordering.

One method to reduce the running time of the new algorithm is to reduce the number of arrays that is generated during the extension phase. However, the number of arrays generated in the extension phase is always at least as large as the number of isomorphism classes. Therefore the number of isomorphism classes to be generated provides a lower bound on the calculation time that can be achieved when reducing the number of arrays generated.

A second way would be to modify the new canonical form and make it stable under extension of the arrays. The new ordering is defined in terms of the delete-one-factor projections and each individual projection depends on the entire array. The author has not found any new ordering that is stable and at the same time efficient to calculate.

Our method is applicable to non-symmetric arrays. However, for non-symmetric arrays columns permutations between columns with a different number of levels are not possible. So, the advantages of the delete-one-factor method for non-symmetric arrays are smaller.

5 Acknowledgements

The author would like to thank Eric Schoen and Peter Goos for reviewing the paper.

References

[1] D. Bulutoglu and F. Margot. Classification of orthogonal arrays by integer programming. *Journal of Statistical Planning and Inference*, 138:654–666, 2008.

[2] J. Chen, D. Sun, and C. Wu. A catalogue of two-level and three-level fractional factorial designs with small runs. *International Statistical Review*, 61:131–145, 1993.

[3] L. Deng and B. Tang. Generalized resolution and minimum aberration criteria for plackett-burman and other nonregular factorial designs. *Statistica Sinica*, 9:1071–1082, 1999.

[4] P. T. Eendebak. Orthogonal array page, 2012. http://www.pietereendebak.nl/oapage/.

[5] P. T. Eendebak. The Orthogonal Array package. Technical report, 2013.

[6] A. Hedayat, N. Sloane, and J. Stufken. *Orthogonal arrays : theory and applications*. Springer, 1999.

[7] B. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[8] B. D. McKay and A. Piperno. Practical graph isomorphism, ii. *CoRR*, abs/1301.1493, 2013.

[9] C. Rao. Factorial experiments derivable from combinatorial arrangements of arrays. *Journal of the Royal Statistical Society Supplement*, 9:128–139, 1947.

[10] K. J. Ryan and D. A. Bulutoglu. Minimum aberration fractional factorial designs with large n. *Technometrics*, 52(2):250–255, 2010.

[11] E. D. Schoen, P. T. Eendebak, and M. V. M. Nguyen. Complete enumeration of pure-level and mixed-level orthogonal arrays. *Journal of Combinatorial Designs*, 18(2):123–140, 2010.

[12] D. Sun, W. Li, and K. Ye. An algorithm for sequentially constructing nonisomorphic orthogonal designs and its applications. Technical report, Department of Applied Mathematics and Statistics, SUNY at Stony Brook, 2002.

[13] H. Xu. Algorithmic construction of efficient fractional factorial designs with large run sizes, 2009.

[14] H. Xu and C. F. J. Wu. Generalized minimum aberration for asymmetrical fractional factorial designs. *Annals of Statistics*, 29:1066–1077, 2001.

A Extension to non-symmetric arrays

The theory in this paper can be extended to include non-symmetric arrays. First we extend the definitions of the distance distribution and GWLPs to non-symmetric arrays. Then we define an ordering on non-symmetric arrays. The rest of the theory is the same as for symmetric arrays.

Let $\mathbf{s} = (s_1, \ldots, s_l)$ with $s_i \geq 2$ and $s_i \neq s_j$ for $i \neq j$. Let $\mathbf{n} = (n_1, \ldots, n_l)$ with $n_i \geq 1$, $\sum_{i=1}^{l} n_i = k$. We use $\mathbf{s^n}$ as a shorthand for $s_1^{n_1} s_2^{n_2} \cdots s_l{}^{n_l}$. For the non-symmetric arrays in $\mathrm{OA}(N; \mathbf{s^n}; t)$ we use the following definitions.

Definition 8 (Distance distribution)**.** Let D be in $\mathrm{OA}(N; \mathbf{s^n}; t)$. Every row $a \in D$ is split as $a = (a_1, \ldots, a_l)$. For $j = (j_1, \ldots, j_l) \in \mathbb{Z}_{n_1} \times \cdots \mathbb{Z}_{n_l}$ we define

$$\begin{aligned} B_j(D) &= B_{j_1, j_2, \ldots, j_l}(D) \\ &= N^{-1} |\{(a, b) : d_H(a_i, b_i) = j_i \text{ for } i = 1, \ldots, l, a \in D, b \in D\}|. \end{aligned}$$

The *MacWilliams transforms* of the distance distribution are defined as

$$B'_j(D) = N^{-1} \sum_{i_1=0}^{n_1} \sum_{i_2=0}^{n_2} \cdots \sum_{i_l=0}^{n_l} B_i(D) P_{j_1}(i_1; n_1, s_1) P_{j_2}(i_2; n_2, s_2) \cdots P_{j_l}(i_l; n_l, s_l)$$

where $P_j(x; k, s) = \sum_{i=0}^{j} (-1)^i (s-1)^{j-i} \binom{x}{i} \binom{k-x}{j-i}$ are the Krawtchouk polynomials.

Definition 9 (Generalized Wordlength Pattern)**.** For an $(N, \mathbf{s^n})$-design D, the generalized wordlength pattern of D is equal to $A(D) = (A_0(D), \ldots, A_k(D))$

with $k = \sum_i n_i$ and

$$A_\iota(D) = \sum_{j_1+j_2+\ldots=\iota} B'_{j_1,\ldots,j_l}(D).$$

Example 4. Let $\mathbf{s} = (4, 2)$, $\mathbf{n} = (2, 1)$. Let D be the orthogonal array in $\mathrm{OA}(16; 4^2 2; 2)$ defined by

$$D = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 3 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 2 & 1 \\ 1 & 3 & 1 \\ 2 & 0 & 1 \\ 2 & 1 & 1 \\ 2 & 2 & 0 \\ 2 & 3 & 0 \\ 3 & 0 & 1 \\ 3 & 1 & 1 \\ 3 & 2 & 0 \\ 3 & 3 & 0 \end{pmatrix}.$$

Then the distance distribution of D is given by (see Definition 8)

$$B_{(0,0)}(D) = 1,\ B_{(1,0)}(D) = 2,\ B_{(2,0)}(D) = 5,$$
$$B_{(0,1)}(D) = 0,\ B_{(1,1)}(D) = 4,\ B_{(2,1)}(D) = 4.$$

The GWLP is given by $A(D) = (1.0, 0.0, 0.0, 1.0)$. ▲

The delete-one-factor projections of a non-symmetric array cannot be com-

pared directly since not all columns have the same number of factor levels. We therefore introduce an ordering that compares the columns only if the column levels are identical. For columns with identical column levels we still use the GWLPs to compare.

Definition 10 (Delete-one-factor ordering for non-symmetric arrays)**.** Let D be in $\mathrm{OA}(N; \mathbf{s^n}; t)$. For each column k we define the mixed projection value $d_{M,k}(D)$ to be the tuple defined by the factor level of column k and the GWLP of the array obtained by deleting column k, so

$$d_{M,k}(D) = (-s_k, d_k(D)).$$

We order the mixed projection values by the usual lexicographic ordering.

Let X and Y be two non-symmetric $N \times n$ arrays. We define X to be smaller than Y in the *delete-one-factor ordering* if there is an l such that $d_{M,j}(X) = d_{M,j}(Y)$ for $j = 1, \ldots, l-1$ and $d_{M,l}(X) < d_{M,l}(Y)$ or $d_{M,j}(X) = d_{M,j}(Y)$ for $j = 1, \ldots, n$ and X is lexicographically smaller than Y.

The definition of $d_{M,k}(D)$ contains a minus sign in front of the factor level to make sure that the arrays in canonical form start with the columns with the highest factor levels. For symmetric arrays the ordering defined above corresponds to the original ordering since all levels s_i are equal.

B Example code

In this section, we give an example of the usage of the Orthogonal Array package [5] to analyse arrays. We calculate the GWLPs of the delete-one-factor arrays and calculate the associated symmetry group.

Example 1: canonical form of an array

```
>>> import oalib
>>> al=oalib.exampleArray(4)
>>> al=oalib.reduceDOPform(al)
>>> al.showarray()
array:
  0   0   0   0   0   0   0
  0   0   0   0   0   0   1
  0   0   0   1   1   1   0
  0   0   0   1   1   1   1
  0   1   1   0   0   1   0
  0   1   1   0   0   1   1
  0   1   1   1   1   0   0
  0   1   1   1   1   0   1
  1   0   1   0   1   0   0
  1   0   1   0   1   1   1
  1   0   1   1   0   0   0
  1   0   1   1   0   1   1
  1   1   0   0   1   0   1
  1   1   0   0   1   1   0
  1   1   0   1   0   0   1
  1   1   0   1   0   1   0
>>> print('GWLP %s' %  str(al.GWLP()) )
GWLP (1.0, 0.0, 0.0, 3.5, 2.5, 0.5, 0.5, 0.0)
>>> for ii in range(0, al.n_columns):
...     bl=al.deleteColumn(ii)
...     print('dof %d: GWLP %s' % (ii, str(bl.GWLP()) ) )
dof 0: GWLP (1.0, 0.0, 0.0, 1.5, 1.0, 0.5, 0.0)
dof 1: GWLP (1.0, 0.0, 0.0, 1.75, 0.75, 0.25, 0.25)
dof 2: GWLP (1.0, 0.0, 0.0, 1.75, 0.75, 0.25, 0.25)
dof 3: GWLP (1.0, 0.0, 0.0, 2.0, 1.0, 0.0, 0.0)
dof 4: GWLP (1.0, 0.0, 0.0, 2.0, 1.0, 0.0, 0.0)
dof 5: GWLP (1.0, 0.0, 0.0, 2.0, 1.0, 0.0, 0.0)
dof 6: GWLP (1.0, 0.0, 0.0, 3.0, 2.0, 0.0, 0.0)
>>> dopgwp = oalib.projectionGWLPvalues ( al )
>>> sg=oalib.symmetry_group(dopgwp, 0)
>>> sg.show(1)
symmetry group: 7 elements, 4 subgroups: 1 2 3 1
```