Eighth International Workshop on Optimal Codes and Related Topics
July 10-14, 2017, Sofia, Bulgaria *pp. 49-54*

A Method for Construction of Orthogonal Arrays [1]

Iliya Bouyukliev iliyab@math.bas.bg
Institute of Mathematics and Informatics, Bulgarian Academy of Sciences
P.O.Box 323, 5000 Veliko Tarnovo, Bulgaria
Maya Hristova maqhristova@gmail.com
Faculty of Mathematics and Informatics, Veliko Tarnovo University,
5000 Veliko Tarnovo, Bulgaria

Abstract. In this article we present a method for generating orthogonal arrays by following the terminology from the book of Kaski and Östergård [5].

1 Introduction

An orthogonal array is a "table" (array) whose entries come from a fixed finite set of symbols, arranged in such a way that there is an integer t so that for every t-columns of the table, all ordered t-tuples of the symbols, appear the same number of times. Orthogonal arrays are of great research interest, as they are widely used in various scientific fields [4].

One of the main tasks associated with orthogonal arrays is their construction and classification. The classification problem is to determine exactly one element from every equivalence class. An essential point is the rejection of isomorphic objects. The methods known so far can be summarized in two groups. The first option is to generate all orthogonal arrays. Then among the generated orthogonal arrays, exhaustive isomorphism testing is performed. The second option is to use isomorph free generation (IF Generation) via canonical form (total lexicographical order).

The methods of the second group are relatively more effective, but finding a total lexicographic order is a computationally difficult task, especially when the orthogonal array has more columns. For this reason, we offer a method for generating orthogonal arrays with isomorph rejection via canonical augmentation. Up to our knowledge, this method has not been developed for such objects (orthogonal arrays) so far.

[1]This research was partially supported by Grant DN 02/2/13.12.2016 of the Bulgarian National Science Fund

2 Main definitions and preliminary results

Let S be a set of s symbols (levels). We will denote these levels by $0, 1, \cdots, s-1$. We interpret the levels as the elements of some special structure such as an alphabet.

Definition 1. *An $N \times k$ array A with entries from S is said to be an orthogonal array with s levels, strength t ($0 \leq t \leq k$) and index λ if every $N \times t$ subarray of A contains each t-tuple based on S exactly λ as a rows. We will denote such orthogonal array by $OA(N, t, s^k)$.*

The parameters N, t, s and k have various names depending on the application of the given orthogonal array. The number of rows N is also called the number of runs or observations. The number of columns k is also known as the number of constraints, different variables or factors whose effect is analyzed. The parameter s specifies the number of symbols, the elements of the orthogonal array. These are the levels of the various factors (their possible values) that affect the final result.

The main properties of the orthogonal arrays are described in details in [4]. We will mention only two which are more important for us: the parameters of an orthogonal array satisfy the equality $\lambda = N/s^t$. Taking the runs in an $OA(N, t, s^k)$ that begin with fixed element (in example 0) and omitting the first column yields an $OA(N/s, t-1, s^{k-1})$.

Definition 2. *Two arrays are said to be isomorphic if one array can be obtained by permuting rows, columns, or factor levels of the other array.*

There are different methods for constructing orthogonal arrays. The isomorph rejection is a key point in solving the task of generating all $OA(N, t, s^k)$. Schoen, Eendebak and Nguyen describe in [7] an algorithm for generating orthogonal arrays with isomorph rejection using a lexicographic order of the arrays. The preliminary isomorphism testing is avoided by retaining only arrays of a specific form (lexicographically minimum in columns).

The technique that is proposed in [3] implements linear programming. Isomorphism testing was carried out with computer-algebraic methods.

For our purposes we represent $OA(N, t, s^k)$ (up to equivalence) as a $N \times k$ matrix A ($a_{i,j} \in S, i = 1, 2, \cdots, N, j = 1, 2, \cdots, k$). We construct the matrix A by columns. Practically, the first two columns depends on the parameters. The main objects for construction a matrix are columns of a given length (N) and they form the set $\mathcal{T}$. The searched matrix is a subset of $\mathcal{T}$. We call *solution* the set of columns that form a matrix with the properties of an orthogonal array. If the matrix A has k columns (k is fixed), A is called *complete solution*. If $i < k$, the solution that we have is called *partial solution*.

Let A is $N \times i$ matrix and we can obtain matrix with $(i+1)$ columns from A. All solutions for column i form set of objects called *children* or possible solution

for A. We denote them by $C(A)$. A is called *parent* for some $B \in C(A)$ and is denoted by $p(A)$. The solutions for column i (the children) are elements from $\mathcal{T}$ which have certain properties (see Definition 1 and Property 2). Let $B \in C(A)$. Then $C(B)$ depends on $C(A)$. We will call them possible solutions for $(i+1)$-th column. We call B fixed solution for i-th column. The fixed solution is called also a real solution (in the next section we denote it by sol_i). It becomes a parent, and with the other possible solutions we are searching for children who have certain properties. In this way we have recursive construction. Since we do not consider all possible column options, but only those that have the properties of an orthogonal array, the search space decreases significantly.

3 Classification algorithm - notation and pseudocode

For the implementation of the algorithm for generating orthogonal arrays that we present, we use algorithmic strategy called backtrack search (backtracking). With this strategy, the complete solution is constructed sequentially. At each step, a set of possible extensions is generated and an attempt is made to extend the current partial solution with the elements of that set. If the current solution can be extended, a step forward is made. Depending on the algorithm's goals, it may stop or step back and continue traversing the search tree.

The backtrack search strategy can be presented as depth-first search of a searched tree [1]. In the search tree, nodes at the first level correspond to subobjects and can be considered as sol_1 solutions for that level. These solutions belong to a finite set $\mathcal{T}$. A solution that corresponds to m can be considered as a m-dimensional vector $(sol_1, sol_2, \cdots, sol_m)$ because it depends directly on the solutions of the previous levels. Thus, at each step we have a set of solutions that have a certain number of children. The solutions at fixed level k correspond to the object we search for. These solutions are called complete.

The solutions are found (by traversing the tree) by dynamic programming (dynamic backtrack search). In our case the node at first level corresponds to an orthogonal array with parameters $OA(N, t, s^2)$ (we use the solution (sol_1, sol_2) as input). We are only interested in a parent's non-isomorphic children. Thus, we use the strategy described in [2]. Each vector is not represented by its elements, but by the number of elements within an interval specified by the parent. Based on the solutions for this interval, we define the possible solutions for the children.

Example 1. *Let an* $OA(N, t, 2^2)$ *be given. Let* $r_1 = N/2$. *The root of the search tree is fixed by this way:* $sol_1 = (\underbrace{11\cdots11}_{r_1}\underbrace{00\cdots00}_{r_1})$. *For* sol_2 *we have* $r_2 = r_1/2$ *and:*

$$sol_2 = (\underbrace{11\cdots11}_{r_2}\underbrace{00\cdots00}_{r_2}\underbrace{11\cdots11}_{r_2}\underbrace{00\cdots00}_{r_2}).$$

The structures of sol_1 and sol_2 defines two intervals of rows for the next column of the orthogonal array - $[0, r_1 - 1]$ and $[r_1, N]$. Each of them has r_2 ones and the same number of zeros.

The backtrack search procedure defines the set of solutions $SOL_{m+1} = SOL_{m+1}(sol_1, sol_2, \cdots, sol_m) \subseteq U, m = 2, \cdots, k$ and for every solution $sol_{m+1} \in SOL_{m+1}$ recursively invokes itself with input $(sol_1, sol_2, \cdots, sol_m, sol_{m+1})$. A solution from level m corresponds to a given column with which we would expand the orthogonal array. If all the elements of the set SOL_{m+1} are considered or the level of the complete solutions is reached $(m = k)$, the return stage in the calling procedure is performed.

To find the solutions from every level m, we use decomposition of integers. Lets consider the sequence of integers $a_1, a_2, \cdots, a_k$. Our goal is to present each of these numbers as the sum of j addends, each of which has a value between 0 and $s(a_i)$ (the size of the corresponding interval), where $i = 1, 2, \cdots, k$. To solve this problem, we use a recursive algorithm with three levels of recursion. In fact, we generate orthogonal arrays whose corresponding matrices are sorted (and part of the redundancy). There may be isomorphic among the objects thus obtained.

The solution obtained in this way is optimal because it always leads us to finding a particular one. On the other hand, not every solution satisfies the condition of an orthogonal array. For this reason, we make additional steps to verify whether the array A obtained after the expansion is orthogonal. The number of these steps is minimized by taking advantage of the specified properties of orthogonal arrays (for arrays with $t \geq 3$).

One of the most time consuming part of the task to finding all non-isomorphic $OA(N, t, s^k)$ is the isomorphism test. The trivial solution to this problem is to generate all orthogonal arrays and then perform exhaustive isomorphism test among them. In practice, this strategy is ineffective because the number of orthogonal arrays could be large. To avoid this problem, we use an isomorph free generation technique.

There are three main approaches for isomorph rejection - with recorded objects, canonical form and canonical augmentation. In our case, we use canonical augmentation. To each orthogonal array, we uniquely juxtapose a binary matrix as follows: to every element $s_i \in s$ we juxtapose vector with length S, $0 \mapsto (10 \cdots 0), 1 \mapsto (010 \cdots 0), \cdots, (s-1) \mapsto (0 \cdots 01)$. The next step is to extend the matrix with n rows, which mark the columns representing one coordinate. Thus, $(N + i)$-th row will have ones in the positions $s(i-1)+1, s(i-1)+2, \cdots, si, i = 1, 2, \cdots, n$ and zeros in the rest. For the canonical augmentation algorithm, we consider binary matrices (EM) instead of matrices of orthogonal arrays (A).

Two orthogonal arrays are isomorphic if their corresponding binary matrices are isomorphic.

Definition 3. *Two binary matrices A and B of the same dimension are isomorphic if one can be obtained from the other by permutations of rows and columns.*

Any permutation of columns that represents the rows of matrix B in rows of the same matrix is called automorphism. The set of all automorphisms of B forms a group called automorphism group and it is denoted with $Aut(B)$. The $Aut(B)$ group splits the set of columns of B into disjoint subsets $O_1, O_2, \cdots, O_k$ called orbits. Two elements (columns) are in one orbit if there is automorphism that leads an element from one to the other.

Let denote by Ω the search space - a finite set, which contains all objects (subobjects) that are considered in the search process (in our case every subset of $\mathcal{T}$).

Definition 4. *A canonical representative map is called the function $\rho : \Omega \to \Omega$, which has the properties: for every $X \in \Omega : \rho(X) \cong X$; for every $X, Y \in \Omega$ from $X \cong Y$ follows that $\rho(X) \equiv \rho(Y)$.*

The canonical form of an object X is its image according to the canonical representative $\rho(X)$. The object X is in canonical form if $\rho(X) = X$.

The canonical representation leads to an ordering of the orbits of X(preceding). This allows us to define one of these orbits as special and to determine a parent test (presented in [6]). Let select the last orbit for a special one and let us denote the last column added to the X with $\dot{x}$. We say that $X \cup \dot{x}$ satisfies the parent test if $\dot{x}$ is in the special orbit.

For the presented algorithm we use the following notation: A is matrix that corresponds to the solution so far (matrix of an orthogonal array), EM is extended matrix that is obtained from A, $\mathcal{T}_k$ is the set of solutions so far, $\hat{C}(A)$ is the set of all non-isomorphic children of A that passed the parent test, SOL_{level} are all possible solutions for given level and $level$ - the number of columns of A. This algorithm is effective when A has more columns.

Algorithm

Procedure CanonicalAugmentation(SOL_{level}, $level$, A)
1: **if** level is equal to k **then**
2: $\mathcal{T}_k := \mathcal{T}_k \cup A$;
3: print(A);
4: **end if**
5: $\hat{C}(A) = \emptyset$
6: **if** $level$ is less than k **then**
7: find the set SOL_{level} of all possible solutions
8: **for** every $sol \in SOL_{level}$ **do**
9: set sol as real solution
10: **for each** solution **in** $SOL_{level} \backslash \{sol\}$ **do**
11: find all possible solutions for $level + 1$

12: obtain EM from sol;
13: **if** EM passes the parent test **then**
14: **if** $\nexists B : B \cong sol, B \in \hat{C}(A)$ **then**
16: $\hat{C}(A) = \hat{C}(A) \cup EM$
17: CanonicalAugmentation(A, level + 1, $SOL_{level+1}$);
18: **end if**
19: **end if**
20: **end for**
21: **end for**
22: **end if**
end procedure

Procedure Main
Input: N, k, t, s: global variables (parameters of an orthogonal array)
Output: all non-isomorphic $OA(N, t, s^k)$
1: $level := 2$;
2: obtain SOL_2 and A;
3: CanonicalAugmentation(A, $level$, SOL_2);
end procedure

References

[1] I. Bouyukliev, 'Q - EXTENSION' – strategy in algorithms, in *Proceedings of the International Workshop ACCT, Bansko, Bulgaria, 2000,* 84-89.

[2] I. Bouyukliev, M. Dzhumalieva-Stoeva and V. Monev, Classification of Binary Self-Dual Codes of Length 40, *IEEE Transactions on Information Theory,* vol.61, No. 8, August 2015.

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

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

[5] P. Kaski and P. R. J. Östergård, *Classification Algorithms for Codes and Designs,* Springer-Verlag, Berlin Heidelberg, 2006.

[6] B. D. McKay, Isomorph-free exhaustive generation, *J. Algorithms 26*, (1998), 306-324.

[7] Eric D. Schoen, Pieter T. Eendebak and Man V.M. Nguyen, Complete Enumeration of Pure-Level and Mixed-Level Orthogonal Arrays, *Journal of Combinatorial Designs*,2, 18, 123-140, 2009.