A Post-Processing Algorithm Applied to Reduce the Size of the Covering Arrays of the NIST Repository

Ph.D. Jose Torres-Jimenez

Laboratorio de Tecnologías de Información, CINVESTAV-Tamaulipas km 5.5

September 19, 2013
# Table of Contents

**Introduction**

- Basic Definitions
- CA Repositories
- Construction Methods
- Manipulation of CAs
- Postprocessing NIST Repository
- Conclusions
Nowadays software products are present in almost all human activities, then it is necessary to assure a high level of functionality of the software products.

It is estimated that at least 50% of the cost of developing a new software component is related to the testing process.\(^1\)

One option to test a software component is to use an exhaustive approach, i.e. test all the possible combinations of the input parameters.

Another option is to use combinatorial testing, that guarantees that all the combinations of certain number of parameters is tested exactly or at least certain number of times.\(^2\)

\(^2\)Cohen et al. [3] y Cohen et al. [4].
Table of Contents

Introduction

Basic Definitions

CA Repositories

Construction Methods

Manipulation of CAs

Postprocessing NIST Repository

Conclusions
Basic Definitions
Orthogonal Arrays (OA)

Definition
Let be \( N, t, k, v, \) and \( \lambda \) five positive integers, an orthogonal array \( OA(\lambda; N, t, k, v) \) is an \( N \times k \) array \( A = (a_{i,j}), 0 \leq i \leq N-1, 0 \leq j \leq k-1, \) over \( \mathbb{Z}_v = \{0, 1, \ldots, v-1\} \) with the property that for any \( t \) distinct columns, there are exactly \( \lambda \) rows that take each value of \( \mathbb{Z}_v^t = \{0, 1, \ldots, v-1\}^t \).

- When \( \lambda = 1 \) the OA is of unitary index.
- For OAs of unitary index and \( v \) a primepower there exists an optimal solution \(^3\)

\(^3\)Bush [5]
Figure 1: The orthogonal array OA_2(8; 2, 7, 2).
Basic Definitions
Orthogonal Arrays (OA)

- The case when \( v \) is a prime power and \( v \geq t \), is solved using arithmetic of a Galois Finite Field. The value of each cell of the OA is the evaluation of the number of row in the number of column, the last column is directly the leftmost coefficient of the number of row. This procedure can be implemented using a logaritm table for a Galois Finite Field.\(^4\)
- The case when \( v \) is a prime power and \( v < t \) is solved using the zerosum algorithm.

\(^4\)Torres et al. \([6]\) and Torres et al. \([7]\)
Basic Definitions
Covering Arrays (CA)

The OAs are a good option to test efficiently software components, but they have the drawback that they do not exist for certain combinations of parameters \((\lambda, k, v, t)\). A relaxed version of the OAs are the CAs (each combination must exist at least once). Given the relaxation, they exist for any combination of parameters and they have a lower number of rows.

Definition
Let be \(N, t, k,\) and \(v\) four positive integers, a covering array \(CA(N; t, k, v)^5\) is an \(N \times k\) array \(A = (a_{ij}), 0 \leq i \leq N - 1, 0 \leq j \leq k - 1,\) over \(\mathbb{Z}_v = \{0, 1, \ldots, v - 1\}\) with the property that for any \(t\) distinct columns, at least one row takes each value of \(\mathbb{Z}_v^t = \{0, 1, \ldots, v - 1\}^t.\)

\[^5\text{when the columns have different order it is called a Mixed Covering Array (MCA)}\]
The covering array construction problem (CACP) consists in constructing a covering array \( CA(N; t, k, v) \) given the parameters \( t, k, \) and \( v \) in such a way the number of rows \( N \) of the covering array is minimal. The smallest \( N \) for which a covering array exists is the covering array number (CAN) for the parameters \( t, k, \) and \( v \), and it is denoted by

\[
CAN(t, k, v) = \min \{ N | \exists CA(N; t, k, v) \}. \tag{1}
\]
Basic Definitions
NP-Complete and Search Space for CAs

- Even, there is no proof that CACP belongs to the class of NP-Complete problems, some related problems are NP-Complete. For instance: the proof if it is possible to add a new row that provides at least a certain number of t-wise missing combinations 6

- The search space of the CACP denoted by $S$ satisfies:

$$v^t \leq S \leq {v^k \choose N}$$  \hspace{1cm} (2)
There are reported only a small number of optimal covering arrays:

- The case $\text{CA}(\nu^t; t, \max(\nu, t) + 1, \nu)$ when $\nu$ is a primepower $^7$. The CAs constructed in this way are equivalent to OAs of unitary index.
- The case $(\nu = 2) \land (t = 2)$ where $k \leq \left(\frac{N-1}{2}\right)^8$

In general the determination of the CAN is very difficult and is the theme of a lot of research, but asymptotically (for large $k$)

$$\text{CAN}(t, k, \nu) \approx \nu^t \log(k)$$  \hspace{1cm} (3)

---

$^7$Bush [5]  
$^8$Rényi [9], Katona [10], Kleitman and Spencer [11]
Basic Definitions
Isomorphism in CAs

Given that:

- The position of the rows in a CA is not relevant, all the CAs obtained by permuting the rows of a CA are isomorphic.
- If all the columns in a CA have the same order (alphabet), all the CAs obtained by permuting the columns of a CA are isomorphic.
- The coverage properties of a CA are not affected if the symbols in one column are permuted (for instance all zeros are exchanged with all ones), all the CAs obtained by permuting symbols within columns are isomorphic.

Definition
For a CA there are $N!k!(v!)^k$ isomorphic CAs. $N!$ permutation of rows, $k!$ permutations of columns, and $(v!)^k$ permutations of symbols in the columns.
Given two CAs $A$ and $B$ CA(6; 2, 5, 2) they are isomorphic if one can be constructed from the other using some permutation of rows, permutation of columns, and permutation of symbols. Considering that $\tau$ defines the row permutation, $\pi$ is the column permutation, and $\phi$ defines the symbol permutation. In the next figure we illustrate how to construct $B$ from $A$.

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

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

$\tau = (0 \ 1 \ 2 \ 3 \ 4 \ 5)$
$\pi = (0 \ 1 \ 2 \ 3 \ 4)$
$\phi = (0 \ 0 \ 0 \ 0 \ 0)$
$\tau' = (3 \ 1 \ 2 \ 5 \ 4 \ 0)$
$\pi' = (4 \ 2 \ 0 \ 3 \ 1)$
$\phi' = (0 \ 1 \ 0 \ 1 \ 0)$

Figure 2: Using CA $A$ we obtain CA $B$ using $\tau'$, $\pi'$, and $\phi'$.
Basic Definitions
Redundancy in CAs

The two cases of optimal CAs mentioned previously (OAs with \((\lambda = 1) \land (\text{prime power}(v))\), and \((v = 2) \land (t = 2))\) result in CAs that do not have redundancy. For any other case the possibility that a CA has a lot of redundancy is very high. A redundant element (usually denoted as wildcard) can take any value and the coverage properties of a CA are not affected (usually are represented with the symbol \(*\)).

Definition
One element of a CA is redundant (wildcard) if we can change that element with any value of its alphabet and the coverage properties of the CA are not affected.
For instance the CA $A(11; 3, 5, 2)$ downloaded from the NIST Repository\(^9\) has one row totally redundant. The CA $B$ indicates with * the redundant elements.

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

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

*Figure 3: CA $A$ can be reduced in one row through the detection of redundant symbols shown in CA $B$ as *  

\(^9\)http://math.nist.gov/coveringarrays/ipof/tables/table.3.2.html
Table of Contents

Introduction

Basic Definitions

CA Repositories

Construction Methods

Manipulation of CAs

Postprocessing NIST Repository

Conclusions
There are three main repositories for uniform CAs:

- **NIST Repository**\(^{10}\) that contains explicit CAs constructed using the IPOG-F algorithm for \( (\nu = \{2, \ldots, 6\}) \land (\tau = \{2, \ldots, 6\}) \).  

- **Charles Colbourn Repository** \(^{11}\) that lists only the best-known sizes for CAs \( (\nu = \{2, \ldots, 25\}) \land (\tau = \{2, \ldots, 6\}) \), it does not provide explicit CAs.

- **CinvestavCA Repository** \(^{12}\) currently it provides explicit CAs. It contains good CAs for \( (\nu = \{2, 3\}) \land (\tau = \{2, \ldots, 6\}) \).

\(^{10}\)http://math.nist.gov/coveringarrays/  
\(^{11}\)http://www.public.asu.edu/~ccolbou/src/tabby/catable.html  
\(^{12}\)http://www.tamps.cinvestav.mx/~jtj/authentication.php
Covering Arrays generated by IPOG-F

This page is an appendix to the paper "Refining the In-Parameter-Order Strategy for Constructing Covering Arrays" by M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker and D.R. Kohn (paper in progress, abstract: HTML/PDF). It holds the covering arrays generated by the main algorithm described, IPOG-F. Many values of $t$ and $y$ are represented. Many of these covering arrays are smaller than currently known in literature, as indicated by the graphs shown in the individual pages. All covering arrays are accessible (in a custom format) for research purposes.

Back to main menu

Choose which CA($t,k,y$) to explore:

- CA(2,k,2)
- CA(3,k,2)
- CA(4,k,2)
- CA(5,k,2)
- CA(6,k,2)
- CA(2,k,3)
- CA(3,k,3)
- CA(4,k,3)
- CA(5,k,3)
- CA(6,k,3)
- CA(2,k,4)
- CA(3,k,4)
- CA(4,k,4)
- CA(5,k,4)
- CA(6,k,4)
- CA(2,k,5)
- CA(3,k,5)
- CA(4,k,5)
- CA(5,k,5)
- CA(6,k,5)
- CA(2,k,6)
- CA(3,k,6)
- CA(4,k,6)
- CA(5,k,6)

The CATables are a service of the Mathematical and Computational Sciences Division of the Information Technology Laboratory of the National Institute of Standards and Technology. Development Status: Active Development. We conform to the NIST Privacy Policy.

Last Updated: 2008-04-17

For comments please mail the Covering Arrays Team: coveringarrays at nist dot gov
Table for CA(3,k,3)

Covering array sizes are given for each $k$ listed, where $k \geq 3$ and $k=3$. The arrays, as generated by the IPOG-F program, are given as links to files of a specific format. For comparison purposes, the sizes are graphed as compared to the best known as listed by Charlie Colbourn's CA Tables page (as accessed when this page was last updated).

Change $k$: Decrease or Increase

Change $v$: Decrease or Increase

<table>
<thead>
<tr>
<th>$k$</th>
<th>CA(3,k)</th>
<th>Filesize</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>24</td>
<td>OK</td>
</tr>
<tr>
<td>5</td>
<td>42</td>
<td>OK</td>
</tr>
<tr>
<td>6</td>
<td>40</td>
<td>OK</td>
</tr>
<tr>
<td>7</td>
<td>52</td>
<td>OK</td>
</tr>
<tr>
<td>8</td>
<td>56</td>
<td>OK</td>
</tr>
<tr>
<td>9</td>
<td>62</td>
<td>OK</td>
</tr>
<tr>
<td>10</td>
<td>66</td>
<td>OK</td>
</tr>
<tr>
<td>11</td>
<td>68</td>
<td>OK</td>
</tr>
<tr>
<td>12</td>
<td>71</td>
<td>OK</td>
</tr>
<tr>
<td>13</td>
<td>76</td>
<td>OK</td>
</tr>
<tr>
<td>14</td>
<td>72</td>
<td>OK</td>
</tr>
<tr>
<td>15</td>
<td>80</td>
<td>OK</td>
</tr>
<tr>
<td>16</td>
<td>82</td>
<td>OK</td>
</tr>
<tr>
<td>17</td>
<td>85</td>
<td>OK</td>
</tr>
<tr>
<td>18</td>
<td>88</td>
<td>OK</td>
</tr>
<tr>
<td>19</td>
<td>91</td>
<td>OK</td>
</tr>
<tr>
<td>20</td>
<td>92</td>
<td>OK</td>
</tr>
</tbody>
</table>
Covering Array Tables for $t=2, 3, 4, 5, 6$

These tables summarize results from the literature and new results obtained through constructive and exhaustive searches. The table for $t=2$ contains the best known covering array for each $n$ and $k$ up to $n=200$ and $k=7$. For $t>2$, the lower bounds are given as $n$ and $k$ increase. The upper bounds are given with references.

For $t=2$, the table contains the best known covering array for each $n$ and $k$. The numbers are given in the format $n,k$. The upper bounds for $t>2$ are given in the format $n,k$. For $t=2$, the lower bounds are given as $n$ and $k$ increase. The upper bounds are given with references.

If you are interested in explicit presentations of covering arrays, which are not necessarily the best known, a good place to start is the NIST Covering Array Tables. Some explicit solutions are also available from the NIST Covering Array Tables.
Table for CAN(6,k,10) for k up to 10000

Last Updated Sun Aug 25 00:00:00 MST 2013

Locate the k in the first column that is at least as large as the number of factors in which you are interested. Then let N be the number of rows (not given) in the second column. A CAN(6,k,10) exists according to a construction in the reference cryptically given in the third column. The accompanying graph plots N vertically against log k (base 10).

<table>
<thead>
<tr>
<th>k</th>
<th>N</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1000000</td>
<td>composition</td>
</tr>
<tr>
<td>8</td>
<td>125497</td>
<td>Add a symbol</td>
</tr>
<tr>
<td>9</td>
<td>149171</td>
<td>Add a symbol</td>
</tr>
<tr>
<td>10</td>
<td>1531156</td>
<td>Add a symbol</td>
</tr>
<tr>
<td>11</td>
<td>1712159</td>
<td>Orthogonal array forms</td>
</tr>
<tr>
<td>12</td>
<td>9750029</td>
<td>Add a factor</td>
</tr>
<tr>
<td>13</td>
<td>1401199</td>
<td>Add a factor</td>
</tr>
<tr>
<td>14</td>
<td>4099499</td>
<td>Add a factor</td>
</tr>
<tr>
<td>15</td>
<td>5382109</td>
<td>Martirosyan-Tsai van Tung</td>
</tr>
<tr>
<td>16</td>
<td>5432333</td>
<td>Martirosyan-Tsai van Tung</td>
</tr>
<tr>
<td>17</td>
<td>7731199</td>
<td>Martirosyan-Tsai van Tung</td>
</tr>
<tr>
<td>18</td>
<td>4213999</td>
<td>double Or (Calderon/Esco) form</td>
</tr>
<tr>
<td>19</td>
<td>4306707</td>
<td>Martirosyan-Tsai van Tung</td>
</tr>
<tr>
<td>20</td>
<td>1002707</td>
<td>Add a factor</td>
</tr>
<tr>
<td>21</td>
<td>4205996</td>
<td>Add a factor</td>
</tr>
<tr>
<td>22</td>
<td>10593018</td>
<td>Martirosyan-Tsai van Tung</td>
</tr>
<tr>
<td>23</td>
<td>21891818</td>
<td>Add a factor</td>
</tr>
<tr>
<td>24</td>
<td>22096109</td>
<td>Martirosyan-Tsai van Tung</td>
</tr>
</tbody>
</table>

Graph:

- 6-CAn with 10 symbols

Log(Number of Factors) vs. N

23/72
Authentication - Covering Arrays

For accessing the Covering Arrays Repository, type guest in both Username and Password.

Username: 
Password: 
Submit

Please cite this repository like:

For v=3 cites:

For v=2 cites:

Note: For request an specific CA that is not available with guest account, you can send a request to the indicated e-mail in Contact Info.
Covering Arrays

For reviewing an specific CA or a group of them, you have to indicate the values of the parameters.

\( v \): The level of each factor
\( t \): Coverage of the strength
\( k \): Number of factors or parameters

CA, v, t, k, Find
### Tables of Covering Arrays

Resulting tables for the values \( v = (3), t = (2) \)

<table>
<thead>
<tr>
<th>N</th>
<th>k</th>
<th>t</th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>2</td>
<td>WC</td>
<td></td>
</tr>
</tbody>
</table>
Table of Contents

Introduction
Basic Definitions
CA Repositories

Construction Methods
Manipulation of CAs
Postprocessing NIST Repository

Conclusions
Construction Methods

Due to the difficulty of solving the CACP a number of methods have been developed. The methods can be classified in five main categories:

- **Exact methods.** These methods guarantee to find an optimal CA, but given the exponential search space they are practical only for small CAs.
- **Greedy methods.** These methods do not guarantee to find an optimal CA, but they are very fast and can be used to construct any CA.
- **Metaheuristic methods.** These methods do not guarantee to find an optimal CA, they take more time than the greedy methods and in many cases they give better CAs than the ones obtained using greedy methods.
- **Algebraic methods.** The algebraic methods involved formulas or operations with mathematical objects such as vectors, finite fields, groups or another (usually) small covering arrays.
- **Manipulation methods.** These methods use covering arrays previously constructed to construct new ones, or transform a CA to more suitable CA.
Manipulation of CAs

There are some useful operations that can be applied to a covering array previously constructed. This section describes four of them: maximization of constant rows, optimal reduction, wildcard detection, and fusion.

- The maximization of constant rows enable that the product of CAs, and the powering of CAs produce redundant rows that are eliminated easily.
- The optimal reduction of CAs, enable to construct small CAs taking as input CAs with greater number of rows and columns.
- The wildcard detection of CAs, will detect redundant elements in a CA.
- The fusion of CAs, enable to exploit systematically the wildcards to reduce the size of a CA.
A constant row in a covering array is a row having the same symbol in all its elements. Formally, the $i$-th row of a covering array $A = (a_{i,j})$ of dimensions $N \times k$ is constant if $a_{i,j} = a_{i,0}$ for $j = 1, 2, \ldots, k - 1$.

By means of the three operations that produce isomorphic covering arrays it is possible to arrange the symbols of the covering array in order to make constant some of its rows.

The constant rows are very useful for the methods of multiplication and powering of covering arrays, because if the covering arrays used have constant rows, then it is possible to delete some rows in the resulting covering array.

This problem can be solved in the domain of graphs. For each row a node is created, and for each pair of rows that are distinct (column by column) an edge is created. The problem is converted to the MAXCLIQUE problem.

---

13 Quiz-Ramos [49]
Manipulation of CAs

Shortening of a CA

- Given a covering array $A$ the **Optimal Shortening of Covering ARrays (OSCAR)** problem consists in finding a submatrix $B$ of a determined size such that the number of missing tuples in $B$ is minimized $^{14}$.

- Let be $\delta$ and $\Delta$ two integers such that $0 \leq \delta \leq N - v$, $0 \leq \Delta \leq k - t$, with the condition that at least one of them is greater than zero.

- The OSCAR problem consists in finding a submatrix $B$ of $A$ of size $(N - \delta) \times (k - \Delta)$ such that the number of missing tuples in $B$ is minimal.

- It was proved that the OSCAR problem is NP-Complete by reducing the MAXCOVER problem to the OSCAR problem.

- The search space of the OSCAR problem is $\binom{N}{N-\delta}\binom{k}{k-\Delta}$.

$^{14}$Carrizales-Turrubiates [50]
Figure 4: The deletion of one row and three columns of the covering array CA(6;2,7,2) produces the covering array CA(5;2,4,2).
Manipulation of CAs
Redundant Elements Detection in a CA

Sometimes a covering array has entries that can be freely modified without affecting the coverage properties of the covering array, that is, without affecting the number of missing tuples of the array. These entries are called wildcards and are commonly represented by the symbol \(.\). Figure 5 shows at the left the covering array CA(7; 2, 8, 2), and shows at right the same covering array with the wildcards it contains.

Figure 5: Wildcards in the covering array CA(7; 2, 8, 2).
Wildcard detection is very important for some postoptimization process for covering arrays. A postoptimization process is a process that tries to reduce the number of rows of a given covering array. One of these process that uses wildcards is the method of Nayeri et al. \(^{15}\)

A methodology to maximize the number of wildcards in a covering array \(^{16}\) proposed three main steps:

- to determine the tuples covered only once;
- to determine the unfixed symbols;
- to enumerate all the possible wildcard configurations.

The first and second steps are seen in the next figure. The elements that paticipate in t-wise combinatios covered once are seen in the second matrix as 0 or 1, the unfixed elements are shown in the second matrix as \(U\). The third step can be implemented using a greedy approach or an exact approach, but in any case both approaches tries to minimize the \(U\) elements that become fixed (to some value of \(Z_v\)), and in this way the remaining \(U\) elements are maximized and converted to wildcards (+).

\(^{15}\)Nayeri et al. [51]

\(^{16}\)Gonzalez-Hernandez et al. [52]
Figure 6: The entries of the covering array CA(7; 2, 8, 2) marked with $U$ are the entries that may become a wildcard.
Manipulation of CAs
Fusion of a CA

Colbourn \(^{17}\) defined the fusion of a CA as:

\[
\text{CAN}(t, k, v) \leq \text{CAN}(t, k, v + 1) = \begin{cases} 
3 & \text{if } t = 2, k \leq v + 1, \text{ } v \text{ is a prime power} \\
2 & \text{otherwise} 
\end{cases}
\]

(4)

The basic mechanism of the fusion operator is to obtain from a covering array \(A = \text{CA}(N; t, k, v)\) another covering array \(B = \text{CA}(M; t, k, v - 1)\) of smaller size by replacing the occurrences of the symbol \(v\) in \(A\) for symbols of the set \(\{0, 1, \ldots, v - 2\}\), and by deleting three or two rows according the cases of expression (4).

\(^{17}\)Colbourn [53], and Colbourn et al. [54]
A generalization of the fusion operator to CAs was proposed. The generalization consists in three main components:

- A wildcard detector, that maximizes the wildcards in a CA. The implementation of this component could be greedy or exact.
- A row replacer that exploits the redundant elements to reduce the rows of the CA. This component copies non-wildcard elements to corresponding wildcard elements (in the same column) in order to obtain a redundant row that can be eliminated. This procedure could end when a redundant row is found, or when the best non-redundant row is found (one row that consumes the less number of wildcard elements). In case a redundant row could not be found, a quasi-CA with the minimum missing t-wise combinations is delivered.
- A metaheuristic algorithm that tries to make zero the number of missing tuples. This procedure is run until the number of missing tuples is zero or a maximum time is consumed.

When the order of the input CA is greater than the order of the output CA an order reductor algorithm is used.

\(^{18}\)Rodriguez-Cristerna [55]
Table of Contents

Introduction

Basic Definitions

CA Repositories

Construction Methods

Manipulation of CAs

Postprocessing NIST Repository

Conclusions
Postprocess NIST Repository

- As part of the Automated Combinatorial Testing for Software (ACTS) project at NIST, a repository of covering arrays is publicly available. An important opportunity is the exploitation of the redundancy of coverage that the covering arrays have in order to reduce its size.
- The application of the Generalized fusion operator produces a lot of improvements of the CA of the NIST Repository.
- The NIST repository was processed using the generalized fusion and We have found 349 new upper bounds.
- We have employed more than 1.5 million of hours of the Xiuhcoatl cluster of the CINVESTAV.
- A total of 21,964 CAs were processed and 20,956 were improved.
- An average reduction of 3.85% was attained.
<table>
<thead>
<tr>
<th>Id.</th>
<th>k</th>
<th>IPOG-F FG-F 1</th>
<th>Id.</th>
<th>k</th>
<th>IPOG-F FG-F 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>42</td>
<td>2046 2002 169 85 2739 27 00 210 45 9227 9104</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>43</td>
<td>2070 2022 170 86 2749 27 06 211 46 9320 9176</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>44</td>
<td>2091 2050 171 87 2762 27 28 212 47 9406 9262</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>129</td>
<td>45</td>
<td>2112 2071 172 88 2770 27 36 213 48 9501 9357</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>130</td>
<td>46</td>
<td>2130 2086 173 89 2783 27 47 214 49 9588 9453</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>131</td>
<td>47</td>
<td>2150 2112 174 90 2792 27 61 215 50 9673 9520</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>132</td>
<td>48</td>
<td>2174 2134 175 91 2805 27 72 216 51 9755 9621</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>133</td>
<td>49</td>
<td>2191 2154 176 92 2815 27 83 217 52 9835 9682</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>134</td>
<td>50</td>
<td>2213 2182 177 93 2825 27 98 218 53 9922 9769</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>135</td>
<td>51</td>
<td>2231 2198 178 94 2836 27 97 219 54 9998 9849</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>136</td>
<td>52</td>
<td>2251 2217 179 95 2847 28 13 220 55 10079 9927</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>137</td>
<td>53</td>
<td>2269 2232 180 96 2857 28 19 221 56 10155 10007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>138</td>
<td>54</td>
<td>2289 2246 181 97 2868 28 34 222 57 10232 10082</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>55</td>
<td>2309 2277 182 98 2877 28 38 223 59 10379 10231</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>56</td>
<td>2327 2281 183 99 2885 28 54 224 60 10454 10302</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>57</td>
<td>2342 2304 184 100 2895 28 62 225 62 10590 10442</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>58</td>
<td>2358 2316 185 101 2909 28 71 226 63 10650 10509</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>59</td>
<td>2373 2335 186 102 2920 28 98 Id.</td>
<td>144</td>
<td>60</td>
<td>2394 2347 187 103 2928 28 98 Id.</td>
</tr>
<tr>
<td>145</td>
<td>61</td>
<td>2408 2370 188 104 2938 29 01 227 21 18779 18260</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>146</td>
<td>62</td>
<td>2425 2387 189 105 2945 29 08 228 22 114775 111818</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>63</td>
<td>2440 2397 190 107 2962 29 28 229 23 118587 115802</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>64</td>
<td>2451 2413</td>
<td>149</td>
<td>65</td>
<td>2468 2439 Id.</td>
</tr>
<tr>
<td>150</td>
<td>66</td>
<td>2482 2447 191 26 6957 67 75</td>
<td>151</td>
<td>67</td>
<td>2498 2459 192 27 7116 69 37 Id.</td>
</tr>
<tr>
<td>152</td>
<td>68</td>
<td>2512 2477 193 28 7267 70 88 232 17 40334 38976</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>69</td>
<td>2527 2487 194 29 7414 72 47 233 18 42102 40820</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>154</td>
<td>70</td>
<td>2542 2504 195 30 7555 73 79 234 19 43833 42554</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>71</td>
<td>2555 2518 196 31 7691 75 27 235 20 45425 44224</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>156</td>
<td>72</td>
<td>2573 2531 197 32 7816 76 49 236 21 46970 45784</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>157</td>
<td>73</td>
<td>2584 2546 198 33 7939 77 82 237 22 48479 47352</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>74</td>
<td>2597 2564 199 34 8064 79 07 238 23 49924 48838</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>75</td>
<td>2609 2588 200 35 8183 80 23 239 24 51287 50180</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>76</td>
<td>2625 2593 201 36 8301 81 37 240 25 52604 51505</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>77</td>
<td>2639 2608 202 37 8420 82 59 241 26 53850 52814</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>78</td>
<td>2648 2615 203 38 8530 83 76 242 27 55069 54032</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>79</td>
<td>2661 2625 204 39 8629 84 77 243 28 56225 55275</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>164</td>
<td>80</td>
<td>2673 2635 205 40 8737 85 83 244 29 57363 56353</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>165</td>
<td>81</td>
<td>2686 2651 206 41 8847 86 93 245 30 58468 57503</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>166</td>
<td>82</td>
<td>2700 2668 207 42 8945 87 91 246 31 59529 58576</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table of Contents

Introduction
Basic Definitions
CA Repositories
Construction Methods
Manipulation of CAs
Postprocessing NIST Repository

Conclusions
Conclusions

- The Covering Arrays (CA) represent an excellent option when it is desired to guarantee a certain level of coverage in the testing of a software component and to use a small number of test cases.
- We have presented basic definitions needed to understand much of the research related to covering arrays.
- We have grouped a lot of research reports related to the construction of CAs, the groups were: Exact methods, Greedy Methods, Metaheuristic Methods, and Manipulation Methods.
- The generalized fusion operator is based on the exploitation of the wildcards of a CA in order to reduce the number of rows of the CA.
- We have processed all the NIST CA Repository using more than 1.5 million CPU hours (Xiuhcoatl Cinvestav Cluster) and have found 349 new upper bounds.
- It is important to highlight that the exploitation of the redundancy of coverage in CAs can enable the creation of better CAs (with lower number of rows).
Thanks!!!
References


References II


References III


References IV


References


References VI


References VII


References IX


References XI


References XII


References XIII


References XIV


