Journal of
Applied & Computational Mathematics
ISSN: 2168-9679

Jimenez and Marquez, J Appl Computat Math 2019, 8:2

Research Article Open Access

Uniqueness of Four Covering Arrays Equivalent to Error-Correcting Codes

Jimenez TJ* **and Marquez II**

Cinvestav-Tamaulipas, Information Technology Laboratory, Carretera Victoria-Soto La Marina, 87130 Victoria Tamps, Mexico

Abstract

A covering array CA(N;t,k,v) is an N × k array over v symbols where every N × t subarray contains as a row each t-tuple over v symbols at least once. Two covering arrays are isomorphic of one can be obtained from the other by permutations of rows, columns, and symbols in the columns. Isomorphic covering arrays form equivalence classes in the set of all CA(N;t,k,v). The problem of classifying covering arrays consists in generating one element of each isomorphism class; if there is only one isomorphism class, then CA(N;t,k,v) is unique. This work introduces two parallel versions of a previously reported algorithm to classify covering arrays. By using these algorithms we determine the uniqueness of the covering arrays CA(32;4,13,2), CA(64;5,14,2), CA(128;6,15,2), and CA(256;7,16,2). We also find that these four covering arrays are equivalent respectively to the unique error-correcting codes (13,32,6), (14,64,6), (15,128,6), and (16,256,6), where (n,M,d) denotes a code with word length n, M code words, and minimum distance d.

Keywords: Covering arrays; Classification of covering arrays; Parallel algorithms; Error-correcting codes

Introduction

A covering array CA(N;t,k,v) is an array of size $N \times k$ where every subarray of t columns contains as a row each t-tuple over Z_v={0,1,...,v-1} at least once [1]. The parameter t is known as the strength of the covering array. A subarray of t columns covers a t-tuple x if the subarray contains x as a row. A covering array CA(N;t,k,v) is optimal if N is the minimum number of rows needed to cover at least once each t-tuple over Z_v in every subarray of t columns [2]. This minimum number of rows is the covering array number of t, k, v, and it is denoted by CAN(t,k,v) [3].

There are three isomorphisms in covering arrays: (a) permutation of rows, (b) permutation of columns, and (c) permutation of symbols in a column. Any combination of these three operations produces an isomorphic covering array; therefore, there are $N!k!(v!)^k$ covering arrays isomorphic to a CA(N;t,k,v) [4]. On the other hand, non-isomorphic covering arrays cannot be transformed among them by permutations of rows, columns, and symbols. A symbol permutation in a column is called a *relabeling* of the column [5].

For particular values of N, t, k, v, the set of all CA(N;t,k,v) is partitioned in classes C_0, C_1, ..., C_{n-1} of isomorphic covering arrays [6-9]. In every isomorphism class C_0, C_1,...,C_{n-1} we select one specific covering array, the canonical one, to be the representative of the class. For X=CA(N;t,k,v) let $\lambda(X)$ be the vector of length N · k that is obtained by arranging the elements of X in column-major order. The array X is canonical if for all Y isomorphic to X the vector $\lambda(X)$ is smaller than or equal to $\lambda(Y)$ in lexicographic order [10-12].

For example, there are three isomorphism classes in the set of all CA(6;2,7,2), and the following three covering arrays CA(6;2,7,2) are the canonical representatives of the classes:

$$\begin{pmatrix} 0&0&0&0&0&0&0\\ 0&0&0&1&1&1&1\\ 0&1&1&0&0&1&1\\ 0&1&1&1&1&0&0\\ 1&0&1&0&1&0&1\\ 1&1&0&1&0&1&0 \end{pmatrix} \quad \begin{pmatrix} 0&0&0&0&0&0&0\\ 0&0&0&0&1&1&1\\ 0&1&1&1&0&0&0\\ 1&0&1&1&0&1&1\\ 1&1&0&1&1&0&1\\ 1&1&1&0&1&1&0 \end{pmatrix} \quad \begin{pmatrix} 0&0&0&0&0&0&0\\ 0&0&0&0&1&1&1\\ 0&1&1&1&0&0&1\\ 1&0&1&1&0&1&0\\ 1&1&0&1&1&0&1\\ 1&1&1&0&1&1&0 \end{pmatrix}$$

Let A be the first of these three covering arrays. Then, $\lambda(A)$=(0,0,0,0,1,1,0,0,1,1,0,1,0,0,1,1,1,0,0,1,0,1,0,1,0,1,0,1,1,0,0,1,1,0,0,1,0,1,1,0,1,0) [13]. For each of the $6!7!(2!)^7$ covering arrays B=CA(6;2,7,2) isomorphic to A, the vector $\lambda(A)$ is smaller than or equal to $\lambda(B)$ in lexicographic order.

The classification of covering arrays consists in generating one element of every isomorphism class [14]. If there are n isomorphism classes we say there are n non-isomorphic CA(N;t,k,v). Some works addressing the classification problem [15]. A study [16] proved that the covering arrays CA(32;4,13,2), CA(64;5,14,2), CA(128;6,15,2), and CA(256;7,16,2) are optimal. However, to the best of our knowledge, the number of isomorphism classes for these four covering arrays is unknown. By using two parallel versions of the algorithm reported earlier [17], we found that there is only one isomorphism class for each of these covering arrays. Therefore, the covering arrays CA(32;4,13,2), CA(64;5,14,2), CA(128;6,15,2), and CA(256;7,16,2) are optimal and unique.

An (n,M,d) binary code is a set of M vectors of length n over Z_2, called codewords, whose minimum mutual distance is d, that is, any two distinct codewords differ in at least d entries, and there is at least one pair of codewords that differ in exactly d entries. If any linear combination of codewords is also a codeword then the code is linear, otherwise the code is nonlinear. A code with minimum distance d can correct [(d-1)/2] or fewer errors. An (n,M,d) code is optimal if M is the maximum number of codewords with length n having minimum distance d.

In this work we found that the nonlinear codes (13,32,6), (14,64,6), (15,128,6), (16,256,6) are equivalent respectively to the covering arrays CA(32;4,13,2), CA(64;5,14,2), CA(128;6,15,2), CA(256;7,16,2). These four codes are known to be optimal and unique; their optimality was proven in a study [1], the uniqueness of the first three codes was proven

***Corresponding author:** Jimenez TJ, 2CINVESTAV-Tamaulipas, Information Technology Laboratory, Carretera Victoria-Soto La Marina, 87130 Victoria Tamps, Mexico, Tel: 52(834)1070220; E-mail: jtj@cinvestav.mx

Received February 20, 2019; **Accepted** March 20, 2019; **Published** March 28, 2019

Citation: Jimenez TJ, Marquez II (2019) Uniqueness of Four Covering Arrays Equivalent to Error-Correcting Codes. J Appl Computat Math 8: 439.

in a study [6], and the uniqueness of the last one was shown in a study [16]. Therefore, any CA(32;4,13,2) is a (13,32,6) code, and reciprocally any (13,32,6) code is a CA(32;4,13,2); and the same applies for the other three pairs of covering arrays and codes: CA(64;5,14,2) and (14,64,6), CA(128;6,15,2) and (15,128,6), and CA(256;7,16,2) and (16,256,6).

The rest of the document is organized as follows: Section 2 presents the parallel algorithms to classify covering arrays; Section 3 describes the computational experimentation to determine the uniqueness of CA(32;4,13,2), CA(64;5,14,2), CA(128;6,15,2), CA(256;7,16,2); Section 4 shows the equivalence of these covering arrays with the codes (13,32,6), (14,64,6), (15,128,6), (16,256,6); and Section 5 contains the conclusions of the work.

Parallel Classification Algorithms

This section describes the sequential classification algorithm introduced in a study [9], and develops to parallel versions of it.

Previous sequential algorithm

The sequential algorithm of a study [9] classifies CA(N;t+1,k+1,v) by testing all possible juxtapositions of v covering arrays A_0=CA(N_0;t,k,v), A_1=CA(N_1;t,k,v),..., $A_{v\text{-}1}$=CA($N_{v\text{-}1}$;t,k,v) where $N=\sum_{i=0}^{v-1} N_i$. When a juxtaposition forms a CA(N;t+1,k,v), a column formed by N_i elements equal to i for $0 \le i \le v$-1 is added to the CA to obtain a CA(N;t+1,k+1,v) as illustrated in the following diagram:

$$CA(N;t+1,k+1,v)=\begin{pmatrix} A_0 & 0 \\ A_1 & 1 \\ \vdots & \vdots \\ A_{v-1} & v-1 \end{pmatrix}$$

Since the algorithm explores all possible ways of constructing CA(N;t+1,k+1,v), the constructed covering arrays cover all isomorphism classes. None of the constructed CA(N;t+1,k+1,v) is canonical, and there may be isomorphic arrays in the results. To obtain only the canonical representatives of the isomorphism classes, the constructed covering arrays are canonized and duplicates are removed.

The algorithm begins by determining the multisets S_j={N_0,N_1,...,$N_{v\text{-}1}$} of v elements such that $N_i \ge$ CAN(t,k,v) and $N=\sum_{i=0}^{v-1} N_i$; these multisets give the allowed number of rows for the covering arrays A_0,A_1,...,$A_{v\text{-}1}$ to be juxtaposed. For each S_j={N_0,N_1,...,$N_{v\text{-}1}$}, the algorithm constructs v sets D_0,D_1,...,$D_{v\text{-}1}$, where for $0 \le i \le v$-1 the set D_i contains the non-isomorphic covering arrays CA(Ni;t,k,v) having N_i rows. For example, for the instance CA(29;3,5,3) the algorithm checks all juxtapositions of three covering arrays with strength two, four columns, and number of rows given by S_0={9,9,11}, or by S_1={9,10,10}. For the multiset S_0={9,9,11} the sets D_0 and D_1 will contain the non-isomorphic CA(9;2,4,3), and the set D_2 will contain the non-isomorphic CA(11;2,4,3).

The sets D_0,D_1,...,$D_{v\text{-}1}$ are used to form a set $P_j=D_0 \times D_1 \times \cdots \times D_{v\text{-}1}$= {($A_0$,$A_1$,...,$A_{v\text{-}1}$): $A_i \in D_i$ for $0 \le i \le$ v-1)} which contains all ways of combining the non-isomorphic covering arrays with number of rows given by S_j. Finally, for each tuple T=(A_0,A_1,...,Av-1) $\in P_j$ there are generated all arrays J=[A_0;$A'_{1,...,}A'_{v\text{-}1}$] where A_0 is placed unchanged and for1 $\le i \le v$-1 the array A'_i is derived from the non-isomorphic A_i=CA(N_i;t,k,v) by a combination of a column permutation and a symbol permutation in each of the k columns. Here [A_0, A'_1,…., $A'_{v\text{-}1}$] denotes the vertical juxtaposition of the arrays A_0, A'_1,...,$A'_{v\text{-}1}$. If J is a CA(N;t+1,k,v), then a column E=(0 1 ··· v-1)^T formed by N_i elements equal to i for $0 \le i \le v$-1 is added to J to form a covering array (JE)=CA(N;t+1,k+1,v).

In this way, from the non-isomorphic covering arrays in T there are generated all possible juxtapositions of v covering arrays that are isomorphic to the v covering arrays in T. Since this is done for each T in Pj, and for each Pj, we have that all possible juxtapositions of v covering arrays of strength t and k columns are explored to see which of them form covering arrays of strength t+1.

Algorithms 1 and 2 taken from a study [9] implement the steps that were described in the previous paragraph. The add column() function constructs the k!(v!)^k arrays A'_i that can be derived from A_i by permutations of columns and symbols in the following way: on every call, the columns of A_i not currently copied to a column of A'_i are copied one a time to column r of A'_i, but each column is copied v! times, one for each symbol permutation. The add column() function fills the column 0 of the blocks A'_1,...,$A'_{v\text{-}1}$, then the function fills the column 1 of the same blocks,and so on; in this way, the array J is constructed one column at a time. However, if the current partial array J with $r<k$ columns is not a covering array of strength t+1, then the remaining columns of the arrays A''_1,...,$A'_{v\text{-}1}$ are not filled, and so not all k!(v!)k arrays A'_i isomorphic to A_i are generated.

First parallel version

The first parallel approach is to parallelize the for-each loop located at line 7 of Algorithm 1. For each multiset {N_0,N_1,...,$N_{v\text{-}1}$} the body of this loop is executed $\prod_{i=0}^{v-1} |D_i|$ times, where $|D_i|$ is the number of non-isomorphic CA(N_i;t,k,v). If the number of multisets S_j={N_0,N_1,...,$N_{v\text{-}1}$}, or the number of non-isomorphic CA(N_i;t,k,v) in the sets D_i is large, then the body of the for-each loop is executed many times. The tuples T=(A_0,A_1,...,$A_{v\text{-}1}$)$\in P_j$ can be processed independently of each other; so a first parallel approach is to divide the processing of the tuples T among all available processors.

Algorithm 1: juxtapose_algorithm(N, k', t', v) [9]

```
1  k ← k' − 1;   t ← t' − 1;   R ← ∅;
2  S ← all multisets {N_0, N_1, ..., N_{v−1}} such that N_i ≥ CAN(t, k, v) and N = Σ_{i=0}^{v−1} N_i;
3  foreach S ∈ S do
4      for i = 0, ..., v − 1 do
5          D_i ← all non-isomorphic CA(N_i; t, k, v);
6      P = D_0 × D_1 × ··· × D_{v−1} = {(A_0, A_1, ..., A_{v−1}) : A_i ∈ D_i for 0 ≤ i ≤ v − 1)};
7      foreach T = (A_0, A_1, ..., A_{v−1}) ∈ P do
8          J ← array(N, k);
9          copy A_0 to the first N_0 rows and k columns of J;
10         add_column(1, 0);
11 canonize the covering arrays CA(N; t + 1, k + 1, v) in R and remove duplicates;
```

Algorithm 2: add_column(i, r) [9]

```
1  foreach column j of A_i not currently assigned to a column of block A'_i of J do
2      foreach permutation ϵ of the symbols {0, 1, ..., v − 1} do
3          copy column j of A_i to column r of A'_i and permute its symbols using ϵ;
4          if i = v − 1 then
5              if r < t or is_covering_array(J, r) = true then
6                  if r = k − 1 then
7                      if (JE) ∉ R then R ← R ∪ {(JE)};
8                  else add_column(1, r + 1);
9          else add_column(i + 1, r);
```

This parallel version is implemented in MPI using the master-slaves model. The master process executes the operations shown in Algorithm 1, and sends the tuples T to the slaves, which execute the operations of Algorithm 2. A tuple T is sent to a slave as soon as the slave finishes the processing of a previous tuple; so the partitioning of the tuples T among the slaves is not static. The CA(N;t+1,k+1,v) constructed by the

slaves are sent to the master process, which stores the received covering arrays in the set *R*.

When all tuples *T* have been processed, the master sends the covering arrays CA(*N*;*t*+1,*k*+1,*v*) to the slaves to be canonized. Slaves canonize the covering arrays CA(*N*;*t*+1,*k*+1,*v*) and return them back to the master process; finally, the master eliminates duplicate covering arrays. A very useful strategy for saving time at the canonization stage is to delete all the arrays, except one, which are equal after a row sorting.

Second parallel version

The second parallelization approach is intended for cases where the number of tuples $T=(A_0,A_1,\ldots,A_{v-1})\in P_j$ is small, but the covering arrays in the tuples T are of considerable size. There may be cases where the number of tuples *T* is just one, but the time required to process the tuple is long. In these scenarios we want to parallelize the work of the add column () function.

As mentioned before, the arrays J that are generated from a tuple $T=(A_0, A_1,\ldots, A_{v-1})$ are those arrays $J=[A_0;A'_1,\ldots, A'_{v-1}]$ where A_0 is fixed and for $1 \le i \le v$-1 the array A'_i i is derived from A_i by permutations of columns and symbols. Then, for the array A_1 we need to generate all $k!\cdot(v!)$k permutations of columns and symbols in the worst case, although many of them are skipped when the algorithm finds that they do not have possibilities of being a CA of strength *t*+1.

The strategy we follow is to assign, or to make fixed, the first FIXED<*k* columns of the first block A'_1 of the array *J*; this block is used to contain covering arrays isomorphic to A_1. Let *P*(*n*,*r*) be the number of permutations of size *r* from *n* objects. We can partition the k! column permutations of A_1 in P(*k*,FIXED) chunks, where the chunks correspond to the P(*k*,FIXED) possible ways to assign the first FIXED columns of A'_1 with FIXED columns of A_1. In addition, each of these chunks is relabeled using the (*v*!)FIXED possible combinations of the *v*! symbol relabelings for each of the FIXED columns. Thus, the number of partitions for a given value of FIXED is $P(k,\text{FIXED})\cdot(v!)^{\text{FIXED}}$.

In the add column() function, that is executed by the slave processes, the first FIXED columns of the block A'_1 are not modified. The other columns of the block are filled in the normal way, that is, any non-fixed column gets in order the columns of A_1 not currently assigned to a column of A'_1, but each column of A_1 is copied *v*! times, one with a distinct relabeling. As in the first parallel version, the constructed CA(*N*;*t*+1,*k*+1,*v*) are sent to the master process. At the end, the covering arrays CA(*N*;*t*+1,*k*+1,*v*) are canonized and duplicated arrays are removed.

Computational Results

Now we extend the computational search done in a study [9] to determine the number of isomorphism classes in the set of all CA(32;4,13,2). In the next Theorem 3.0.1 we prove that there is only one non-isomorphic CA(32;4,13,2).

Theorem 3.0.1: CA(32;4,13,2) is unique.

Proof: CA(32;4,13,2) is constructed by juxtaposing CA(N_0;3,12,2) and CA(N_1;3,12,2) where $N_0+N_1=32$, and by adding to the juxtaposition a column formed by N_0 zeros and N_1 ones. Given that CAN (3,12,2)=15 [3], the multisets to consider are $\{N_0=15,N_1=17\}$ and $\{N_0=16,N_1=16\}$. The NonIsoCA algorithm of a study [8] gives 2 non-isomorphic CA(15;3,12,2), 44,291 non-isomorphic CA(16;3,12,2), and 1,091,971,630 non-isomorphic CA(17;3,12,2). So, the number of tuples *T*=(A0,A1) to process in Algorithm 1 is (2) (1,091,971,630)+44,2912. Because this number of tuples *T* is large, we use the first parallel version of the algorithm. None juxtaposition of A_0=CA(15;3,12,2) and A_1=CA(17;3,12,2) produced a CA(32;4,13,2). In addition, all CA(32;4,13,2) that were produced by juxtaposing two CA(15;3,12,2) belongs to the same isomorphism class. Then, there is only one non-isomorphic CA(32;4,13,2).

In this way, CA(32;4,13,2) is both optimal and unique. Figure 1 shows the canonical representative of the unique isomorphism class of all CA(32;4,13,2). The following Theorem 3.0.2 establishes the uniqueness of the covering arrays CA(64;5,14,2), CA(128;6,15,2), and CA(256;7,16,2).

Theorem 3.0.2: CA(64;5,14,2), CA(128;6,15,2), and CA(256;7,16,2) are unique.

Proof: The juxtapositions of the unique CA(32;4,13,2) with itself generate only one non-isomorphic CA(64; 5,14,2). Similarly, from the juxtapositions of the unique CA(64;5,14,2) with itself we obtain only one CA(128;6,15,2). Finally, the juxtapositions of the unique CA(128;6,15,2) with itself produce only one non-isomorphic CA(256;7,16,2).

Therefore, the covering arrays CA(64;5,14,2), CA(128;6,15,2), and CA(256;7,16,2) are optimal and unique. In the three cases of Theorem 3.0.2 we used the second version of the parallel algorithm with value FIXED=3; so the number of partitions was $\frac{k!}{(k-3)!}.2^3$. The classification of these covering arrays was possible tanks to the fact that there is only one tuple $T=(A_0,A_1)$ to be processed in each case. Also the strategy of eliminating the CA(*N*;*t*+1,*k*+1,*v*) that are equal after a row sorting helps a lot in the classification of these three covering arrays. For example, the canonization of a CA(256;7,16,2) takes more than 24 hours, but all CA(256;7,16,2) that were constructed by the slave processes were identical after a row sorting; so it was not necessary to canonize the constructed CA(256;7,16,2) since there was only one.

To the best of our knowledge, the four covering arrays that were classified in this section are the larger ones that has been classified by a computational method. In particular CA(256;7,16,2) is very large to be processed by an algorithm based on constructing covering arrays cell by cell. The algorithm parallelized in this work has the advantage of constructing the covering arrays subcolumn by subcolumn, where the

$$\begin{pmatrix}
0000000000000000111111111111111\\
0000000011111111000000001111111\\
0000011001111110011111100000011\\
0001110101000111010001110001101\\
0010110101001011100110010110011\\
0011011010010011010110101001011\\
0011101010100011101000110011101\\
0100111010001101011010010110100\\
0101011001101010100101011010011\\
0101100110011010101011000011010\\
0110010110100110100010111101000\\
0110101001010110010011010110010\\
0111000101110001011100010111000
\end{pmatrix}^{T}$$

Figure 1: Canonical representative of the unique isomorphism class of CA (32; 4; 13; 2) (shown transposed).

subcolumns used are not arbitrary but columns of covering arrays with one unit less of strength.

Equivalence with Error-Correcting Codes

An orthogonal array $OA_\lambda(N;t,k,v)$ is an $N \times k$ array where every subarray of t columns covers exactly λ times each t-tuple over Z_v. Usually the value of λ is not specified because it can be derived from the other parameters. The nonlinear codes (13,32,6), (14,64,6), (15,128,6), and (16,256,6) are known to be equivalent to the orthogonal arrays OA8(32;2,13,2), OA8(64;3,14,2), OA8(128;4,15,2), and OA8(256;5,17,2) respectively. In fact, the equivalence between codes and orthogonal arrays is well-known: to a (k,M,d) code with dual distance $d^\perp$ corresponds an $OA(N;d^\perp\text{-}1,k,2)$; see chapters 4 and 5 of a study [7]. However, in this work we found that the above mentioned codes are also equivalent to the covering arrays CA(32;4,13,2), CA(64;5,14,2), CA(128;6,15,2), and CA(256;7,16,2). These covering arrays have a larger strength than their equivalent orthogonal arrays.

The covering array CA(32;4,13,2) has the same size as the (13,32,6) code resulting from extending the (12,32,5) code constructed by Nadler [12]. So, we verify if the rows of the covering array have minimum distance 6, and the answer was positive; both objects have 384 pairs of rows or codewords with distance 6, 48 pairs with distance 8, and 64 pairs with distance 10. In addition, we validated by computer that the extended Nadler code is a covering array of strength four. Because the extended Nadler code is optimal [1] and unique [6], any CA(32;4,13,2) is equivalent to the extended Nadler code, and reciprocally any extended Nadler code is a CA(32;4,13,2). The isomorphisms of codes are the same as those of covering arrays: permutation of codewords, permutation of coordinates (columns), and permutations of symbols in a coordinate.

Similarly, we found that the rows of CA(64;5,14,2), CA(128;6,15,2), and CA(256;7,16,2) have minimum distance 6; so they are equivalent respectively to the (14,64,6), (15,128,6), and (16,256,6) codes. These three codes are optimal [1] and unique; the uniqueness of the first two is proved in a study [6], and the uniqueness of (16,256,6) is proved in a study [16]. Thus, there is a correspondence between (14,64,6) and CA(64;5,14,2), between (15,128,6) and CA(128;6,15,2), and between (16,256,6) and CA(256;7,16,2).

The code (16,256,6) is the well-known code resulting from extending the (15,256,5) code discovered by Nordstrom and Robinson [13], and independently by Semakov and Zinoviev [15]. For the (16,256,6) code several constructions exist, among which are the construction of Semakov and Zinoviev [15] from permutation matrices; constructions from the binary Golay code [5,14]; the construction of Liu, et al. [11] from two linear (8,16,4) codes; and the construction of Forney, et al. [4] as the binary image of the octacode.

Now we construct a particular (16,256,6) code, which is the canonical representative of all covering arrays CA(256;7,16,2), by juxtaposing eight extended Nadler codes, or covering arrays CA(32;4,13,2). The juxtaposition of the eight CA(32;4,13,2) forms an array of size 256 × 13; the other 3 columns required to have an array of size 256 × 16 are formed by eight blocks, where for $0 \le i \le 7$ the i-th block contains 32 times the 3-tuple that is the binary representation of i, as shown in Figure 2a. The eight CA(32;4,13,2) are constructed as follows: for each tuple (a,b,c) of Figure 2b add the corresponding vector of length 13 to the rows of the canonical CA(32;4,13,2) of Figure 1, and place the resulting array in the block of Figure 2a associated to the block with 32 occurrences of the tuple (a,b,c). After placing the eight copies of CA(32;4,13,2), sort the rows of the entire array of size 256 × 16. The result is the canonical representative of the unique isomorphism class of all CA(256;7,16,2). From the canonical CA(256;7,16,2), the canonical CA(128;6,15,2) is obtained by taking the first 128 rows and deleting the first column. Similarly, the canonical CA(64;5,14,2) is obtained from the canonical CA(256;7,16,2) by taking the first 64 rows and deleting the first 2 columns.

As said before, to obtain the correspondence between the (16,256,6) code and the covering array CA(256;7,16,2) we computed the minimum distance among the rows of CA(256;7,16,2) and obtained d=6; and given the uniqueness of the (16,256,6) code we concluded that both objects are the same. On the other hand, from the structure of the (16,256,6) code show in Figure 2a we can obtain that the (16,256,6) code is a covering array of strength seven. At the beginning of the section we mentioned that the (16,256,6) code is an $OA_8(256;5,16,2)$; then, in any subarray of five columns the binary tuples of length 5 are covered eight times, and so in any subarray of three columns the binary tuples of length 3 are covered 32 times. Let S be a subarray of three columns of the (16,256,6) code. Sort the rows of the code so that the rows of the subarray S are sorted in non-decreasing order. The other 13 columns of the sorted array can be partitioned into 8 blocks of size 32 × 13. Each of these eight blocks must have minimum distance 6, because the other 3 columns contain the same 3-tuple. However, there is only one array of size 32 × 13 with minimum distance 6, and it is the covering array CA(32;4,13,2).

(a)

(0, 0, 0)	CA(32; 4, 13, 2)
(0, 0, 1)	CA(32; 4, 13, 2)
(0, 1, 0)	CA(32; 4, 13, 2)
(0, 1, 1)	CA(32; 4, 13, 2)
(1, 0, 0)	CA(32; 4, 13, 2)
(1, 0, 1)	CA(32; 4, 13, 2)
(1, 1, 0)	CA(32; 4, 13, 2)
(1, 1, 1)	CA(32; 4, 13, 2)

(b)

Tuple	Vector
(0, 0, 0)	(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
(0, 0, 1)	(0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0)
(0, 1, 0)	(0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1)
(0, 1, 1)	(0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1)
(1, 0, 0)	(0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1)
(1, 0, 1)	(0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1)
(1, 1, 0)	(0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0)
(1, 1, 1)	(0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0)

Figure 2: Construction of the (16; 256; 6) code from 8 copies of the extended Nadler code. (a) The partitioning of the (16; 256; 6) code. (b) The vectors added to the extended Nadler code of Figure 1 to obtain the 8 copies required in (a).

Conclusion

In this work we determined the uniqueness of the optimal covering arrays CA(32;4,13,2), CA(64;5,14,2), CA(128;6,15,2), and CA(256;7,16,2). These results were obtained by using two parallel versions of a previously reported algorithm based on juxtaposing v covering arrays of strength t and k columns to construct covering arrays of strength t+1 and k+1 columns. To the best of our knowledge, these four covering arrays are the larger ones that has been classified by a computational method.

We also found that the above four covering arrays are equivalent respectively to the optimal and unique error-correcting codes (13,32,6), (14,64,6), (15,128,6), and (16,256,6). Because of the uniqueness of both type of objects there is a correspondence between the following four pairs: CA(32;4,13,2) and (13,32,6), CA(64;5,14,2) and (14,64,6), CA(128;6,15,2) and (15,128,6), CA(256;7,16,2) and (16,256,6). The optimality of both kind of objects is interesting because in covering arrays the optimality is related to the minimum number of rows, and in codes the optimality has to do with the maximum number of codewords.

References

1. Best M, Brouwer A, MacWilliams F, Odlyzko A, Sloane N (1978) Bounds for binary codes of length less than 25. IEEE Transactions on Information Theory 24: 81-93.
2. Choi S, Kim HK, Oh DY (2012) Structures and lower bounds for binary covering arrays. Discrete Mathematics 312: 2958-2968.
3. Colbourn CJ, Keri G, Soriano PPR, Schlage-Puchta JC (2010) Covering and radius-covering arrays: Constructions and classification. Discrete Applied Mathematics 158: 1158-1180.
4. Forney Jr GD, Sloane NJA, Trott MD (1993) The Nordstrom-Robinson code is the binary image of the octacode. In: Proceedings DIMACS/IEEE Workshop on Coding and Quantization, pp: 19-26.
5. Goethals JM (1971) On the Golay perfect binary code, Journal of Combinatorial Theory. Series A, 11: 178-186.
6. Goethals JM (1977) The extended Nadler code is unique (corresp.). IEEE Transactions on Information Theory, 23: 132-135.
7. Hedayat AS, Sloane NJA, Stufken J (1999) Orthogonal Arrays: Theory and Applications. Springer-Verlag, New York, pp: 416.
8. Izquierdo-Marquez I, Torres-Jimenez J (2018) New optimal covering arrays using an orderly algorithm. Discrete Mathematics, Algorithms and Applications 10: 1850011.
9. Izquierdo-Marquez I, Torres-Jimenez J (2019) New covering array numbers. Applied Mathematics and Computation.
10. Kokkala JI (2017) Computational methods for classification of codes.
11. Liu C, Ong B, Ruth G (1973) A construction scheme for linear and non-linear codes. Discrete Mathematics 4: 171-184.
12. Nadler M (1962) A 32-point n=12, d=5 code. IRE Transactions on Information Theory 8: 58-58.
13. Nordstrom AW, Robinson JP (1967) An optimum nonlinear code. Information and Control 11: 613-616.
14. Semakov NV, Zinoviev VA (1969) Balanced codes and tactical configurations. Problems of Information Transmission 5: 22-28.
15. Semakov NV, Zinoviev VA (1969) Complete and quasi-complete balanced codes. Problems of Information Transmission 5: 14-18.
16. Snover SL (1973) The uniqueness of the Nordstrom-Robinson and the Golay binary codes.
17. Torres-Jimenez J, Izquierdo-Marquez I (2016) Construction of non-isomorphic covering arrays. Discrete Mathematics, Algorithms and Applications 8: 1650033.