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(NT,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_2^{v} with t at least once. The parameter t is known as the strength of the covering array. A covering array CA(NT,k,v) is optimal if N is the minimum number of rows needed to cover at least once each t-tuple over Z_2^v in every subarray of t columns. There is a UC problem of classifying covering arrays consisting in generating one element of each isomorphism class; if there is only one isomorphism class, then CA(NT,k,v) is unique. The works introduce 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 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(NT,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_2^v with t at least once [1]. The parameter t is known as the strength of the covering array. A covering array CA(NT,k,v) is optimal if N is the minimum number of rows needed to cover at least once each t-tuple over Z_2^v in every subarray of t columns [2]. There is a UC problem of classifying covering arrays consisting in generating one element of each isomorphism class; if there is only one isomorphism class, then CA(NT,k,v) is unique. The works introduce 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 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.

Abstract
A covering array CA(NT,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_2^v with t 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. Isomorphism covering arrays form equivalence classes in the set of all CA(NT,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(NT,k,v) is unique. The works introduce 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 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(NT,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_2^v with t at least once [1]. The parameter t is known as the strength of the covering array. A covering array CA(NT,k,v) is optimal if N is the minimum number of rows needed to cover at least once each t-tuple over Z_2^v in every subarray of t columns. The parameter t is known as the strength of the covering array. A covering array CA(NT,k,v) is optimal if N is the minimum number of rows needed to cover at least once each t-tuple over Z_2^v in every subarray of t columns. There is a UC problem of classifying covering arrays consisting in generating one element of each isomorphism class; if there is only one isomorphism class, then CA(NT,k,v) is unique. The works introduce 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 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.

Abstract
A covering array CA(NT,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_2^v with t 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(NT,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(NT,k,v) is unique. The works introduce 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 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(NT,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_2^v with t at least once [1]. The parameter t is known as the strength of the covering array. A covering array CA(NT,k,v) is optimal if N is the minimum number of rows needed to cover at least once each t-tuple over Z_2^v in every subarray of t columns. The parameter t is known as the strength of the covering array. A covering array CA(NT,k,v) is optimal if N is the minimum number of rows needed to cover at least once each t-tuple over Z_2^v in every subarray of t columns. There is a UC problem of classifying covering arrays consisting in generating one element of each isomorphism class; if there is only one isomorphism class, then CA(NT,k,v) is unique. The works introduce 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 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.

Abstract
A covering array CA(NT,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_2^v with t 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(NT,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(NT,k,v) is unique. The works introduce 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 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.

Abstract
A covering array CA(NT,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_2^v with t 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(NT,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(NT,k,v) is unique. The works introduce 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 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.

Abstract
A covering array CA(NT,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_2^v with t 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(NT,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(NT,k,v) is unique. The works introduce 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 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.
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 (14,64,4), 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), and CA(256,7,16,2); Section 4 shows the equivalence of these covering arrays with the codes (13,32,6), (14,64,4), (15,128,6), and (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 $A_0, A_1, ..., A_v$, where $N = \sum_{i=0}^{t} N_i$. When a juxtaposition forms a CA(N+t+1,k+1,v), a column formed by $N_i$ elements equal to $i$ for $0 \leq i \leq v-1$ is added to the CA to obtain a CA(N+t+1,k+1,v) as illustrated in the following diagram:

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

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 multiset $S = \{N_0, N_1, ..., N_v\}$ of $v$ elements such that $N_i \geq CAN(t,k,v)$ and $N = \sum_{i=0}^{t} N_i$; these multiset gives the allowed number of rows for the covering arrays $A_0, A_1, ..., A_v$ to be juxtaposed. For each $S = \{N_0, N_1, ..., N_v\}$, the algorithm constructs $v$ sets $D_0, D_1, ..., D_v$, where $0 \leq i \leq v-1$; the set $D$ contains the non-isomorphic covering arrays $CA(N_i,t,k,v)$ 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 $S = [9,9,11]$, or by $S = [9,10,10]$. For the multiset $S = [9,9,11]$ 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 $D_0, D_1, ..., D_v$ are used to form a set $P = D_0 \times D_1 \times ... \times D_v$, where $A \in P$ for $0 \leq i \leq v-1$. Each $A$ contains all ways of combining the non-isomorphic covering arrays with number of rows given by $S$. Finally, for each tuple $A \in P$ there are generated all arrays $J = \{A_0 A', A', \ldots, A_v \}$, where $A_0$ is placed unchanged and for $i \leq i \leq v-1$ the array $A'$ is derived from the non-isomorphic $A \in CA(N_i,t,k,v)$ by a combination of a column permutation and a symbol permutation in the $k$ columns. Here $\{A_0, A', A', \ldots, A_v \}$ denotes the vertical juxtaposition of the arrays $A_0, A', A', \ldots, A_v$. If $J$ is a CA(N+t+1,k,v), then a column $E = (0 \ 1 \ \ldots \ v-1)^T$ formed by $N_i$ elements equal to $i$ for $0 \leq i \leq v-1$ 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 $T_i$ and for each $P_j$, 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!) arrays $A'$ that can be derived from $A$ by permutations of columns and symbols in the following way: on every call, the columns of $A$, not currently copied to a column of $A'$, are copied one a time to column $r$ of $A'$, but each column is copied $v$ times, one for each symbol permutation. The add column() function fills the column 0 of the blocks $A', \ldots, A'_v$, 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 $A', \ldots, A'_v$ are not filled, and so not all k!(v!)k arrays $A'$, isomorphic to $A$, are generated.

**First parallel version**

The first parallel approach is to parallelize the for-loop located at line 7 of Algorithm 1. For each multiset $\{N_0, N_1, ..., N_v\}$ the body of this loop is executed $\prod_{i=0}^{t} D_i$ times, where $|D|$ is the number of non-isomorphic CA(N,i,k,v). If the number of multisets $S = \{N_0, N_1, ..., N_v\}$, or the number of non-isomorphic CA(N,i,k,v) in the sets $D$ is large, then the body of the for-each loop is executed many times. The tuples $T = (A_0 A'_1, \ldots, A'_v) \in \mathbb{P}$ 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 algorithms (N, k, v, r) [9]**

```plaintext
1. for i = 0 to v - 1 do
2.   S = all multisets {N_i} such that N_i \geq CAN(i,k,v) and N = \sum_{i=0}^{t} N_i;
3.   foreach S \in S do
4.     \text{add column}(0, S, E, T, i, k, v);
5.     \text{if } r < k \text{ then do}
6.       \text{if } r < k \text{ then do}
7.         \text{add column}(i, \text{J}, 0);
8.       \text{end add column}(i, \text{J}, 0);
9.     \text{end if } r < k \text{ then do}
10.    \text{if } r < k \text{ then do}
11.       \text{canonize the covering arrays CA(N+t+1,k+1,v) in R and remove duplicates;}
12. \text{end if } r < k \text{ then do}
```

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

```plaintext
1. for each column i of A, not currently assigned to a column of block $R_i$ of J do
2.   \text{copy column } i \text{ of } A \text{ to column } r \text{ of } A' \text{ and permute its symbols using } e;\n3.   \text{if } r < k \text{ then do}
4.     \text{add column}(r, \text{J}, 0);
5.   \text{else add_column}(i, r + 1);
6.   \text{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 masters is the most unique covering array obtained; and if it is not unique, a simple function is used to determine the uniqueness of the covering array.
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 \( CA(N+1,k+1,v) \) to the slaves to be canonized. Slaves canonize the covering arrays \( CA(N+1,k+1,v) \) 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 \( T=(A_o,A_1, \ldots, A_v)EP \) 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 \( T=(A_o,A_1, \ldots, A_v) \) are those arrays \( J=[A_o,A'_1, \ldots, A'_v] \) where \( A_v \) is fixed and for \( 1 \leq i \leq v-1 \) the array \( A'_i \) is derived from \( A_v \) by permutations of columns and symbols. Then, for the array \( A_v \), we need to generate all \( k! \) 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 \( k \) columns of the first block \( A'_1 \) to delete all the arrays, except one, which are equal after a row sorting.

In the add column() function, that is executed by the slave processes, the first \( k \) 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 \( A_o \) not currently assigned to a column of \( A'_1 \), but each column of \( A_v \) is copied \( v! \) times, one with a distinct relabeling. As in the first parallel version, the constructed \( CA(N+1,k+1,v) \) are sent to the master process. At the end, the covering arrays \( CA(N+1,k+1,v) \) 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 \( CA(N;3,12,2) \) and \( CA(N;3,12,2) \) where \( N+N=32 \), 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 \( \left\{N_o=15, N_1=17\right\} \) and \( \left\{N_o=16, N_1=16\right\} \). The NonIsoCA algorithm of a study [8] gives 2 non-isomorphic CA(15,12,2), 4491 non-isomorphic CA(16,12,2), and 1,091,971,630 non-isomorphic CA(17,3,12,2). So, the number of tuples \( T=(A_0,A_1) \) to process in Algorithm 1 is \( 2 \times 1,091,971,630 \) + 44,291.

Because this number of tuples \( T \) is large, we use the first parallel version of the algorithm. None juxtaposition of \( A_i=CA(15,3,12,2) \) and \( A_j=CA(17,3,12,2) \) 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 \( FIXED=3 \). The classification of these covering arrays was possible thanks to the fact that there is only one tuple \( T=(A_o,A_1) \) to be processed in each case. Also the strategy of eliminating the CA(\( N_i;1,k+1,v) \) 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
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\(_N(\lambda; k, v)\) is an \(N \times k\) array where every subarray of \(r\) columns covers exactly \(\lambda\) times each \(t\)-tuple over \(Z\). Usually the value of \(\lambda\) 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 OA\(_8(32;2,13,2)\), OA\(_8(64;3,14,2)\), OA\(_8(128;4,15,2)\), and OA\(_8(256;5,17,2)\) respectively. In fact, the equivalence between codes and orthogonal arrays is well-known; to a \((k,M,\lambda)\) code with dual distance \(d^\perp\) corresponds an OA\(_{(Nk)}^{(M-1)k}\); 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 x 13; the other 3 columns required to have an array of size 256 x 16 are formed by eight blocks, where for 0 ≤ i ≤ 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 x 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\(_{256}(5,16,2)\); 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 x 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 x 13 with minimum distance 6, and it is the covering array CA\(_{32}(4,13,2)\).

![Figure 2](image-url) 

**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