New covering array numbers

Idelfonso Izquierdo-Marquez\textsuperscript{a}, Jose Torres-Jimenez\textsuperscript{a,*}
\textsuperscript{a}CINVESTAV-Tamaulipas, Cd. Victoria, Tamaulipas, M\'exico.

\section*{Abstract}
A covering array $CA(N; t, k, v)$ is an $N \times k$ array on $v$ symbols such that each $N \times t$ subarray contains as a row each $t$-tuple over the $v$ symbols at least once. The minimum $N$ for which a $CA(N; t, k, v)$ exists is called the covering array number of $t, k,$ and $v$, and it is denoted by $CAN(t, k, v)$. We prove that $CA(N; t+1, k+1, v)$ can be obtained from the juxtaposition of $v$ covering arrays $CA(N_0; t, k, v), \ldots, CA(N_v; t, k, v)$, where $N = v^{-1}\sum_{i=0}^{v} N_i$. Given this, we developed an algorithm that constructs all possible juxtapositions and determines the nonexistence of certain covering arrays which allow us to establish the new covering array numbers $CAN(4, 13, 2) = 32$, $CAN(5, 8, 2) = 52$, $CAN(5, 9, 2) = 54$, $CAN(5, 14, 2) = 64$, $CAN(6, 15, 2) = 128$, and $CAN(7, 16, 2) = 256$. Additionally, the computational results are the improvement of the lower bounds of 13 covering array numbers.

\textbf{Keywords:} Covering array number, Juxtaposition of covering arrays, Non-isomorphic covering arrays

\section{Introduction}
Covering arrays (CAs) are combinatorial objects with applications in software and hardware testing. Recently, CAs have been used to detect hardware Trojans [13]. A covering array $CA(N; t, k, v)$ with strength $t$ and order $v$ is an $N \times k$ array over $\mathbb{Z}_v = \{0, 1, \ldots, v-1\}$, such that every subarray formed by $t$ distinct columns contains as a row each $t$-tuple over $\mathbb{Z}_v$ at least once. In testing applications, the columns of the CA represent parameters or inputs for the component under test, and a $CA(N; t, k, v)$ ensures to test all possible combinations of values among any $t$ inputs.

Given $t, k,$ and $v$, the problem of constructing optimal CAs is the problem of determining the minimum $N$ for which a $CA(N; t, k, v)$ exists. This minimum $N$ is called the covering array number (CAN) of $t, k,$ and $v$, and it is denoted by $CAN(t, k, v) = \min\{N : \exists CA(N; t, k, v)\}$. Similarly, another problem is finding the maximum value of $k$ for which a $CA(N; t, k, v)$ exists; this $k$ is denoted by $CAK(N; t, v) = \max\{k : \exists CA(N; t, k, v)\}$. Values CAN and CAK are related: $CAN(t, k, v) = \min\{N : CAK(N; t, v) \geq k\}$ and $CAK(N; t, v) = \max\{k : CAN(t, k, v) \leq N\}$.

The determination of $CAN(t, k, v)$ is a difficult combinatorial problem for general values of $t, k,$ and $v$. Some relevant cases with known values of CAN are the following:

- $CAN(1, k, v) = v$ for each $k \geq 1$. 
• CAN($t, t, v$) = $v^t$ for each $t \geq 1$.
• CAN($t, t + 1, 2$) = $2^t$ for each $t \geq 1$.
• CAN($t, v + 1, v$) = $v^t$ when $v$ is prime power and $v > t$ [2].
• CAN($t, t + 1, v$) = $v^t$ when $v$ is prime power and $v \leq t$ [6].
• CAN($3, v + 2, v$) = $v^3$ when $v = 2^n$ [2].
• CAN($v - 1, v + 2, v$) = $v^{v-1}$ when $v = 2^n$ [8] (Corollary 3.8).
• CAN($2, k, 2$) = $N$, where $N$ is the least positive integer satisfying $\binom{N - 1}{\frac{N - 1}{2}} \geq k$ [12, 14].
• CAN($t, t + 2, 2$) = $l^4 2J$ for each $t \geq 1$ [11].

Apart from these cases only a few CANs have been found. Some optimal CAs for $2 \leq v \leq 8$ are listed in [7], and other CANs were determined by computational search in [10, 15].

There are two main ways to find covering array numbers: by combinatorial analysis and by computational search. In the first case there are the works mentioned in the list above. In the second case there are the algorithms that explore the entire search space to determine the existence or nonexistence of CAs. The use of these algorithms is limited to small values of $N$, $t$, $k$, $v$, because the size of the search space grows exponentially. A first approximation of the size of the search space is $v^{Nk}$, which is the number of $N \times k$ matrices over $\mathbb{Z}_v$. Of course, some matrices, like the zero matrix, do not have possibilities of being a CA of strength $t$. Although, limited to small cases, computational algorithms are very promising to find CANs for general values of $t$, $k$, and $v$.

To improve the performance of search algorithms, non-isomorphic exploration of the search space should be used. One class of these algorithms is the orderly algorithm. To implement an orderly algorithm to construct CAs it is necessary to identify the three isomorphisms that exist for CAs: row permutations, column permutations, and symbol permutations in a column. Then, any combination of row, column, and symbol permutations produces an equivalent CA. In the search process only one equivalent CA must be explored. Some algorithms that take advantage of the isomorphisms in CAs are [20, 9, 1, 10].

The present work addresses the task of finding exact values of CAN($t, k, v$), i.e. optimal CAs, by means of computational search. The strategy of our algorithm is significantly different from previously reported strategies. Instead of constructing the target covering array, say CA($N; t+1, k+1, v$), from scratch or directly in the search space for $N$, $t+1$, $k+1$, $v$, our algorithm constructs CA($N; t+1, k+1, v$) by juxtaposing $v$ CAs of strength $t$ and $k$ columns CA($N_0; t, k, v$), CA($N_1; t, k, v$), $\ldots$, CA($N_{v-1}; t, k, v$), where $N = \sum_{i=0}^{v-1} N_i$. This greatly reduces the candidate arrays for the $v$ blocks needed to form a CA($N; t+1, k+1, v$), and a direct implication is that the explored search space is reduced in many cases.

The above searching algorithm is used to determine the existence or nonexistence of CAs. From the nonexistence of certain CAs, we can derive the optimality of others. The main results reported in this paper are the following covering array numbers: CAN($4, 13, 2$) = 32, CAN($5, 8, 2$) = 52, CAN($5, 9, 2$) = 54, CAN($5, 14, 2$) = 64, CAN($6, 15, 2$) = 128, and CAN($7, 16, 2$) = 256. To the best of our knowledge, these CANs had not been determined before. Additional results are the improvement of the lower bounds of CAN($5, 10, 2$), CAN($5, 11, 2$), CAN($5, 12, 2$), CAN($5, 13, 2$), CAN($6, 9, 2$), CAN($7, 10, 2$), CAN($8, 11, 2$),
CAN(9, 12, 2), CAN(10, 13, 2), CAN(11, 14, 2), CAN(3, 7, 3), CAN(3, 9, 3), and CAN(4, 7, 3).
The remainder of the document is organized as follows: Section 2 gives more details about isomorphic and non-isomorphic CAs; Section 3 presents the algorithm to determine the existence of a CA with strength $t + 1$ and $k + 1$ columns from the juxtaposition of $v$ CAs with strength $t$ and $k$ columns; Section 4 shows an implementation of the crucial step of the algorithm, which is the generation of all possible juxtapositions derived from $v$ non-isomorphic CAs; Section 5 describes the executions of our algorithm to obtain the main results of the work; finally, Section 6 presents the conclusions.

2. Isomorphic and non-isomorphic CAs

There are three symmetries in CAs: permutations of rows, permutations of columns, and permutations of symbols in a column. These operations do not change the coverage properties of a CA, and the CAs obtained by combining these operations are isomorphic to the initial CA. Then, two covering arrays $A$ and $B$ are isomorphic (denoted by $A \sim \cdots \sim B$) if $A$ can be derived from $B$ by permutations of rows, columns, and/or symbols in the columns. A covering array $A = CA(N; t, k, v)$ has $N! k! (v!)^k$ isomorphic CAs, including itself, because there are $N!$ possible row permutations, $k!$ possible column permutations, and $(v!)^k$ possible combinations of symbol permutations in the $k$ columns of $A$. A symbol permutation in a column is also called column relabeling.

On the other hand, two CAs $A$ and $B$ are non-isomorphic if it is not possible to derive $A$ from $B$ by permutations of rows, columns, and symbols. Non-isomorphic CAs are truly distinct CAs. In this work the terms “non-isomorphic” and “distinct” will be used interchangeably when refered to CAs.

The set of all CAs with the same parameters $N$, $t$, $k$, $v$, can be partitioned in classes of isomorphic CAs. Thus, the relation of being isomorphic is an equivalence relation in the set of all CAs with the same parameters $N$, $t$, $k$, $v$. CAs in the same class are equivalent, but sometimes it is convenient to take the smallest CA in lexicographic order as the representative of the class. We define the minimum CA of an isomorphism class as the CA with the smallest lexicographic order when its $N \cdot k$ elements are arranged in column-major order.

The NonIsoCA algorithm of [10] generates the minimum CA of every isomorphism class. All CAs generated by the algorithm are non-isomorphic among them, since they belong to distinct classes. Therefore, we refer to the minimum CAs of the isomorphism classes as the non-isomorphic CAs. If there are $r$ isomorphism classes for some parameters $N$, $t$, $k$, $v$, then we say there are $r$ non-isomorphic $CA(N; t, k, v)$. The NonIsoCA algorithm will be used to generate the non-isomorphic CAs required by our searching algorithm presented in Section 3.

Additionally, we adapted the NonIsoCA algorithm of [10] to receive as input the set of all non-isomorphic CAs with $k$ columns and to produce the non-isomorphic CAs with $k + 1$ columns. We call this adaptation of the NonIsoCA algorithm ExtendNonIsoCA. The ExtendNonIsoCA algorithm extends a partial subarray with $k$ columns with columns that: are greater than the last one in lexicographic order; form a CA of strength $t$ with the current $k$ columns; and the CA with $k + 1$ columns is the minimum of its isomorphism class.

3. Existence of CAs

In this work, the existence or the nonexistence of $CA(N; t + 1, k + 1, v)$ is determined by checking all possible ways of juxtaposing vertically $v$ CAs with order $v$, strength $t$, and $k$ columns, and by adding to this
juxtaposition a column formed by v column vectors of constant elements. Determining the existence or the nonexistence of CAs is key to find new covering array numbers. The strategy to determine the existence of CAs is based on the following theorem:

**Theorem 1.** CA($N$; $t + 1, k + 1, v$) exists if and only if there exist v covering arrays CA($N$; $t, k, v$), CA($N_1$; $t, k, v$), . . . , CA($N_{v−1}$; $t, k, v$), where $N = \sum_{i=0}^{v−1} N_i$, that juxtaposed vertically form a CA($N$; $t + 1, k, v$).

**Proof.** Assume $C = CA(N; t + 1, k + 1, v)$ exists. For $0 \leq i \leq v − 1$ let $N_i$ be the number of elements equal to $i$ in the last column of $C$. Construct $C^t$ isomorphic to $C$ by reordering the rows of $C$ in such a way the elements of the last column of $C^t$ are sorted in non-decreasing order. For $0 \leq i \leq v − 1$ let $B_i$ be the block of the $N_i$ rows of $C^t$ where the symbol in the last column is $i$. Divide $B_i$ in two blocks: $A_i$ containing the first $k$ columns and $I$ containing the last column; then $C^t$ has the following structure:

$$
C^t = \begin{pmatrix}
A_0 & 0 \\
\vdots & \vdots \\
A_{v−1} & v − 1
\end{pmatrix}
$$

The juxtaposition of blocks $A_0, A_1, \ldots, A_{v−1}$ form a CA($N$; $t + k, v$) because $C^t$ has strength $t + 1$; then, to complete the first part of the proof we need to show that blocks $A_0, A_1, \ldots, A_{v−1}$ are CAs of strength $t$. The indices of the columns are 0-based, so the last column of $C^t = CA(N; t + 1, k + 1, v)$ has index $k$. Any combination ($c_0, c_1, \ldots, c_i = k$) of $t + 1$ columns containing the last column of $C^t$ covers in block $B_i$ all $(t + 1)$-tuples of the form $(x_0, x_1, \ldots, x_{t−1}, x_t = i)$ over $\mathbb{Z}_v$. Thus, every combination of $t$ columns ($c_0, c_1, \ldots, c_{t−1}$) from the first $k$ columns of $C^t$ covers all $t$-tuples $(x_0, x_1, \ldots, x_{t−1})$ over $\mathbb{Z}_v$ in every block $A_i$, and therefore $A_i = CA(N_i; t, k, v)$.

Now, suppose there are $v$ covering arrays $A_0 = CA(N_0; t, k, v), A_1 = CA(N_1; t, k, v), \ldots, A_{v−1} = CA(N_{v−1}; t, k, v)$, whose vertical juxtaposition forms $G = CA(N; t + 1, k, v)$ of strength $t + 1$, where $N = \sum_{i=0}^{v−1} N_i$. Let $E = (0 \ 1 \ \cdots \ v − 1)^T$ be the column formed by concatenating vertically $N_i$ elements equal to $i$ for $0 \leq i \leq v − 1$. Because every $A_i$ is a CA of strength $t$, we have that for $0 \leq i \leq v − 1$ any submatrix formed by $t$ columns of $A_i$ joined with $I$, covers all $(t + 1)$-tuples of the form $(x_0, x_1, \ldots, x_{t−1}, x_t = i)$. Then, any summatry formed by $t$ columns of $G$ and by column $E$ covers all $(t + 1)$-tuples over $\mathbb{Z}_v$, and therefore, the horizontal concatenation of $G$ and $E$ is a CA($N; t + 1, k, v$).

By Theorem 1 if CA($N; t + 1, k + 1, v$) exists, then it can be constructed by juxtaposing vertically $v$ CAs with strength $t$ and $k$ columns. Also, if CA($N; t + 1, k + 1, v$) does not exist, there are no v CAs with strength $t$ and $k$ columns that juxtaposed vertically form a CA($N; t + 1, k, v$).

The algorithm developed in this work to determine the existence of CA($N; t + 1, k + 1, v$) verifies all possible juxtapositions of v CAs with strength $t$ and $k$ columns to see if one of them produces a CA($N; t + 1, k, v$). In a negative case, CA($N; t + 1, k + 1, v$) does not exist; and in a positive case CA($N; t + 1, k, v$) exists and the algorithm generates all non-isomorphic CA($N; t + 1, k, v$).

The first step of the algorithm is to determine the multisets $S_j = \{N_0, N_1, \ldots, N_{v−1}\}$ of $v$ elements such that $N_j \geq CA(N; t, k, v)$ and $N = \sum_{j=0}^{v−1} N_j$. These multisets will be called valid multisets. To determine, for example if CA(27; 3, 5, 3) exists, we need to check all juxtapositions of three CAs with strength two, four columns, and number of rows given by $S_0 = \{9, 9, 9\}$. In this case $S_0 = \{9, 9, 9\}$ is the unique valid multiset because $CA(2, 4, 3) = 9$. Therefore, if CA(27; 3, 5, 3) exists, it is necessarily composed by three CA(9; 2, 4, 3). On the other hand, to determine if CA(29; 3, 5, 3) exists, the multisets to consider are $S_0 = \{9, 9, 11\}$ and $S_1 = \{9, 10, 10\}$, because CA(29; 3, 5, 3) can be composed by two CAs of nine rows and one CA of eleven rows, or by one CA of nine rows and two CAs of ten rows.
The second step of the algorithm is to generate the non-isomorphic CAs (i.e., the minimum CAs of every isomorphism class) with strength $t$, $k$ columns, and number of rows given by a valid multiset $S_j = \{N_0, N_1, \ldots, N_{v-1}\}$. From each non-isomorphic CA, the other members of its isomorphism class will be derived by permutations of rows, columns, and symbols. To construct the non-isomorphic CAs, we can use the NonIsoCA algorithm of [10], or any other algorithm for the same purpose.

Given a valid multiset $S_j = \{N_0, N_1, \ldots, N_{v-1}\}$, let $D_i$ $(0 \leq i \leq v - 1)$ be the set of all non-isomorphic CA($N; t, k, v$). From the CAs in the sets $D_i$, all juxtapositions of $v$ CAs whose number of rows are given by $S_j$, will be generated. In the example with $S_0 = \{9, 9, 11\}$, the sets $D_0$ and $D_1$ contain the non-isomorphic CA($9; 2, 4, 3$), and the set $D_2$ contains the non-isomorphic CA($11; 2, 4, 3$). Now, let $P_J = \{(A_0, A_1, \ldots, A_{v-1}) : A_i \in D_i \text{ for } 0 \leq i \leq v - 1\}$ be the Cartesian product of the sets $D_i$; then $P_J$ contains all possible ways to combine the non-isomorphic CAs with number of rows given by $S_j$.

The next step of the algorithm is to check all juxtapositions derived from a tuple of $P_J$. For a tuple $T = (A_0, A_1, \ldots, A_{v-1}) \in P_J$, let $[A_0; A_1; \cdots; A_{v-1}]$ denote the juxtaposition of the $v$ CAs in $T$, using this array, all arrays $J = [A^t_{r_0}; A^t_{r_1}; \cdots; A^t_{r_{v-1}}]$ are generated. Each $A^t_{r_v}$ is derived from exactly one $A_t \in T$ by permutations of rows, columns, and symbols in the columns; in other words, indices $r_0, r_1, \ldots, r_{v-1}$ are a permutation $\pi$ of $(0, 1, \ldots, v - 1)$, and $A^t_{r_\pi(i)}$ is isomorphic to $A^t_{r_{\pi^{-1}}(i) \cdot \cdots \cdot i \cdot \cdots \cdot (v - 1)!}$. Each array $J$ is checked to see if it is a CA($N; t + 1, k, v$). If this is the case, then CA($N; t + 1, k + 1, v$) exists by Theorem 1, and the CA is obtained by adding to $J$ the column $E = (0 \ 1 \ \cdots \ v - 1)^T$.

For each tuple of $P_J$, all possible arrays $J$ are generated; then, all possible juxtapositions of $v$ CAs with strength $t$, $k$ columns, and number of rows given by a valid multiset $S_j$ are explored. Since this is done for every valid multiset $S_j$, we have that all possible juxtapositions of $v$ CAs with strength $t$ and $k$ columns are explored.

The number of juxtapositions verified to determine the existence of CA($N; t + 1, k + 1, v$) may be very large, however some juxtapositions produce isomorphic arrays, and to accelerate the search we need to skip as many isomorphic arrays as possible. Fortunately, the number of arrays $J$ created for a tuple $T = (A_0, A_1, \ldots, A_{v-1})$ of $P_J$ can be reduced considerably. Consider the horizontal concatenation of an array $J$ and the column $E = (0 \ 1 \ \cdots \ v - 1)^T$, denoted as ($JE$):

$A^t_{r_0}$

$JJE = A^t_{r_1}$

\vdots

$A^t_{r_{v-1}} \ v - 1$

We can reorder the rows of ($JE$) in such a way that the array derived from $A_0$ is placed in the first rows of ($JE$), the array derived from $A_1$ is placed next, and so on. This row permutation produces an array ($JE'$) isomorphic to ($JE$), and by a permutation of symbols in the last column of ($JE'$), it is possible to transform the last column of ($JE'$) in $(0 \ 1 \ \cdots \ v - 1)^T$:...
\[ A_{r_0} \quad 0 \quad A_0 \quad 0 \]
\[ \vdots \quad \vdots \quad \vdots \quad \vdots \]
\[ (JE) = \quad A_{r_1} \quad \ldots \quad A_1 \quad \ldots \quad A_1 \]
\[ \vdots \quad \vdots \quad \vdots \quad \vdots \]
\[ A_{v_{v-1}} \quad v - 1 \quad A_{v_{v-1}} \quad 1_{v-1} \quad A_{v_{v-1}} \quad v - 1 \]

Therefore, the arrays \( J \) generated from a tuple \( T = (A_0, A_1, \ldots, A_{v-1}) \) of \( P_J \) are those arrays \( J = [A_0^i; A_1^i; \ldots; A_{v-1}^i] \) where for \( 0 \leq i \leq v - 1 \) the array \( A_i^j \) is derived from \( A_i \) by permutations of rows, columns, and symbols. So, the number of arrays \( J \) is reduced from \( v! \prod_{k=0}^{v-1} N_k!k!(v!)^k \) to \( \prod_{k=0}^{v-1} N_k!k!(v!)^k \).

Another reduction in the number of arrays \( J = [A_0^i; A_1^i; \ldots; A_{v-1}^i] \) is possible: we can permute the first \( N_0 \) rows of \( J \), and permute the columns and symbols in the entire array \( J \) to get an array \( J^F = [A_0^{i_0}; A_1^{i_1}; \ldots; A_{v-1}^{i_{v-1}}] \), where \( A_0^i = A_0 \) and \( A_1^i, \ldots, A_{v-1}^i \) are another CAs isomorphic to the original arrays \( A_1, \ldots, A_{v-1} \). Thus, the following arrays are isomorphic:

\[ A_0^i \quad 0 \quad A_0 \quad 0 \]
\[ A_1^i \quad 1 \quad A_1^i \quad 1 \]
\[ \vdots \quad \vdots \quad \vdots \quad \vdots \]
\[ A_{v_{v-1}}^i \quad v - 1 \quad A_{v_{v-1}}^i \quad v - 1 \]

In this way, the arrays \( J \) to be generated from a tuple \( T = (A_0, A_1, \ldots, A_{v-1}) \) of \( P_J \) are those arrays \( J = [A_0; A_1; \ldots; A_{v-1}] \) where \( A_0 \) is fixed, and for \( 1 \leq i \leq v - 1 \) the array \( A_i^j \) is derived from \( A_i \) by permutations of rows, columns, and symbols. Since \( A_0 \) is fixed, the number of arrays \( J \) is now \( \prod_{k=0}^{v-1} N_k!k!(v!)^k \).

However, note that block \( A_i^j \) of \( J \) is completed with a column vector \( i \) formed by \( N_i \) elements equal to \( i \). Then, any row permutation of \( A_i^j \) produces an array isomorphic to \( J \). On the contrary, column and symbol permutations in \( A_i^j \) do not always produce arrays isomorphic to \( J \). Therefore, the only arrays \( J \) that is necessary to explore are those arrays where for \( 1 \leq i \leq v - 1 \) the block \( A_i^j \) is derived from \( A_i \) by permutations of columns and symbols. In this way, the number of arrays \( J \) to be generated from a tuple \( T \in P_J \) is \( [k!(v!)^k]^{v-1} \).

The Algorithm 1 determines the existence of CAs. Given the input parameters \( k \) and \( t \) the algorithm obtains the number of columns \( k = t^2 - 1 \) and the strength \( t = t^2 - 1 \) of the CAs to be juxtaposed. The generation of the valid multisets \( \{N_0, N_1, \ldots, N_{v-1}\} \) requires the value of \( \text{CAN}(t, k, v) \). The key function is \textit{generate juxtapositions} \((T)\), where arrays \( J = [A_0^i; A_1^i; \ldots; A_{v-1}^i] \) are generated from a tuple \( T \) of the set \( P \) ; this function will be described in Section 4.

For each \( C = \text{CA}(N; t + 1, k + 1, v) \) constructed by \textit{generate juxtapositions}(), we obtain the minimum array \( C^* \) of the isomorphism class to which \( C \) belongs, and then \( C^* \) is added to the set \( R \) of distinct CAs. To obtain \( C^* \), we assume the existence of a function \textit{minimum}() of the function \textit{is minimum}(X, r) of [10].

### 4. Generation of Juxtapositions
The crucial step of the algorithm to determine the existence of $CA(N; t + 1, k + 1, v)$ is to generate all arrays $J = [A_0; A_1^t; \cdots; A_{v-1}^t]$ from a given $v$-tuple of CAs $T = (A_0, A_1, \ldots, A_{v-1})$. Recall that in each $J$, ...
Algorithm 1: construct($N$, $k^t$, $t$, $v$)

1. $k \leftarrow k^t - 1$;
2. $t \leftarrow t - 1$;
3. $R \leftarrow \emptyset$;
4. $S \leftarrow \text{all multisets } \{N_0, N_1, \ldots, N_{v-1}\}$ such that $N_i \geq \text{CAN}(t, k, v)$ and $N = \sum_{i=0}^{v-1} N_i$;
5. foreach $S \in S$ do
6.   for $i = 0, \ldots, v - 1$ do
7.     $P = D_0 \times D_1 \times \cdots \times D_{v-1} = \{(A_0, A_1, \ldots, A_{v-1}) : A_i \in D_i \text{ for } 0 \leq i \leq v - 1\}$;
8.     foreach $T = (A_0, A_1, \ldots, A_{v-1}) \in P$ do
9.       generate/juxtapositions($T$);
10.  if $R = \emptyset$ then
11.     CA($N; t+1, k+1, v$) does not exist;
12.  else
13.     CA($N; t+1, k+1, v$) exists and $R$ contains the minimum of each isomorphism class;

the array $A_0$ is fixed, and for $1 \leq i \leq v - 1$ the array $A_i^t$ is derived from $A_i$ by permutations of columns and symbols. After generating an array $J$, the algorithm checks if $J$ is a CA($N; t+1, k, v$).

This section presents an algorithm that performs this crucial step. The algorithm constructs $J$ one column at a time validating that each new column forms a CA of strength $t + 1$ with the columns previously added to $J$. This is done to avoid the exploration of arrays $J$ with no possibilities of being a CA($N; t+1, k, v$). The algorithm starts by constructing the following array $J$, in which block $A_0$ is fixed, and blocks $F_1, \ldots, F_{v-1}$ are unassigned or free; later on, these arrays will be filled with arrays derived from $A_1, \ldots, A_{v-1}$ by permutations of columns and symbols:

\[
\begin{array}{c}
A_0 \\
F_1 \\
\vdots \\
F_{v-1}
\end{array}
\]

For $1 \leq i \leq v - 1$ let $f_{i0}, f_{i1}, \ldots, f_{ik_{i-1}}$ be the $k$ columns of $F_i$, and let $a_{i0}, a_{i1}, \ldots, a_{ik_{i-1}}$ be the $k$ columns of $A_i$. Then, the previous array $J$ is equivalent to this one:

\[
\begin{array}{cccc}
a_{i0} & a_{i1} & \cdots & a_{ik_{i-1}} \\
f_{i0} & f_{i1} & \cdots & f_{ik_{i-1}} \\
\vdots & \ddots & \ddots & \ddots \\
f_{iv-10} & f_{iv-11} & \cdots & f_{iv-1k_{i-1}}
\end{array}
\]

The algorithm fills column 0 in all free blocks, then it fills column 1 in all free blocks, and so on. In this way, columns $f_{i0}, f_{i2}, \ldots, f_{iv-1}$ are filled first, then columns $f_{i1}, f_{i3}, \ldots, f_{iv-1}$, are filled, and so on. When the first $t + 1$ columns of all free blocks have been filled or assigned, the algorithm checks if they form a
CA of strength $t + 1$. Columns are indexed from 0, so the first $t + 1$ columns of $J$ are formed by  columns
If the first \( t + 1 \) columns of \( J \) form a CA of strength \( t + 1 \), then the algorithm advances to the next column of the free blocks, and column \( f_{t+1} \) is assigned, then column \( f_{2t+1} \) is assigned, and so on until column \( f_{t+v-1} \) is assigned. At this point, the algorithm verifies if the current first \( t + 2 \) columns of \( J \) form a CA of strength \( t + 1 \). In the negative case, the current value of \( f_{t+v-1} \) is replaced by its next available value to see if now the first \( t + 2 \) columns of \( J \) form a CA of strength \( t + 1 \). This is done for all available values of \( f_{t+v-1} \), and when all values are checked, the algorithm backtracks to \( f_{t+v-2} \), and assigns to it its next available value; in the next step the algorithm advances to \( f_{t+v-1} \) to check again all its available values.

To construct all possible arrays \( J \), the algorithm fills the free block \( F_1 \) with all isomorphic CAs derived from \( A_i \) by permutations of columns and symbols. Thus, the possible values for a column of \( F_i \) are the columns obtained by permuting symbols in the columns of \( A_i \); so, the number of available values for a column of \( F_i \) is \((v!)^k \). When the first \( r \) columns of \( F_i \) have been assigned, the number of available values for \( f_r \) is \((v!)^k \) which are the \( v! \) symbol permutations of the columns of \( A_i \) not assigned to one of the first \( r \) columns of \( F_i \).

In each free block \( F_i \) the algorithm works as follows: columns of \( A_i \) are added to \( F_i \) in such a way \( f_0 \) gets all columns \( a_0, \ldots, a_{t-1} \) in order; then for a fixed value of \( f_0 \), column \( f_i \) gets in order all columns of \( F_i \) distinct to the one assigned to \( f_0 \); and for fixed values of \( f_0 \) and \( f_i \), column \( f_2 \) gets in order all columns of \( A_i \) not currently assigned to \( f_0 \) or \( f_i \); the same applies for the other columns of \( F_i \). In this way \( F_i \) gets all CAs derived from \( A_i \) by permutation of columns.

However, for each permutation of columns of \( A_i \) the algorithm of Section 3 requires to test all possible symbol permutations in the columns of \( A_i \). Symbol permutations are integrated in the following way: suppose the first \( r \) columns of \( F_i \) have been assigned, and suppose the next free column \( f_r \) of \( F_i \) gets assigned column \( a_{ij} \) of \( A_i \); we can consider that the current value of \( f_r \) is the identity symbol permutation of \( a_{ij} \); the next \( v! - 1 \) values to assign to \( f_r \) are the other \( v! - 1 \) symbol permutations of \( a_{ij} \). When all symbol permutations of \( a_{ij} \) are assigned to \( f_r \), the next value for \( f_r \) is the identity symbol permutation of the next column of \( A_i \) that has not been assigned to \( f_r \).

The function \texttt{generate juxtapositions()} of Algorithm 2 receives as parameter a \( v \)-tuple of CAs. The purpose of this function is to initialize the fixed block \( A_0 \) of \( J \), and to initialize with \texttt{FALSE} the elements of a \( v \times k \) matrix called \texttt{assigned}; this matrix is used to record which columns of \( A_i \) are currently assigned to a column of \( F_i \). The last sentence of the function \texttt{generate juxtapositions()} is a call to the function \texttt{add column()}, which fills the free blocks \( F_i \) with CAs derived from \( A_i \) by permutations of columns and symbols. The function \texttt{add column()} is called from \texttt{generate juxtapositions()} with arguments 1 and 0, because the first column to fill in array \( J \) is column 0 of the free block \( F_1 \), or \( f_{i_0} \).

The function \texttt{add column()} of Algorithm 3 receives as parameters an index \( i \) of a free block and an index \( r \) of a column of the free block; the work of the function is to set column \( f_i \). The variable \( F_i \) indicates the block of \( J \) in which a CA derived from \( A_i \) will be placed. The function \texttt{add column()} relies on recursion to assign column 0 of every free block, then to assign column 1 of every free block, and so on. Additionally, in every free block \( F_i \), recursion allows to test in column \( r \) all columns of \( A_i \) not assigned to a column of \( F_i \); the main \texttt{for} loop iterates over all columns \( j \) of \( A_i \), but the body of the loop is executed only for those columns \( j \) for which \texttt{assigned}[i][j] is equal to \texttt{FALSE}. Recursion also allows to check in order the \( v! \) symbol permutations \( E_{0}, E_{1}, \ldots, E_{v!-1} \) of the column of \( A_i \) assigned to column \( r \) of \( F_i \). If a CA(\( N; t + 1, k, v \)) is
constructed, then column $E$ is appended to it to form $C = \text{CA}(N; t + 1, k + 1, v)$. The function \text{minimum()} obtains $C^*$, and finally this last array is added to the set $R$ of non-isomorphic $\text{CA}(N; t + 1, k + 1, v)$.

For a $v$-tuple of CAs $T = (A_0, A_1, \ldots, A_{v-1})$, Algorithm 2 and its helper function Algorithm 3, generate in the worst case $k!(v!)^{v-1}$ arrays $J = [A_0; A_1; \cdots; A_{v-1}]$, since array $A_0$ is fixed and $A_1, \ldots, A_{v-1}$ are derived respectively from $A_1, \ldots, A_{v-1}$ by permutations of columns and symbols. However, the condition that every new column added to the partial array $J$ must form a CA of strength $t + 1$ with the previous columns of $J$ reduces the number of arrays $J$ explored. For example, if the condition fails at the column with index $j$, then in each free block $F_i$ we skip the remaining $(k - j - 1)!$ permutations of columns for the free columns $f_{i+1}, \ldots, f_{i-1}$, plus the $(v!)^{k-j-1}$ associated permutations of symbols.

As We can see in Algorithm 3, what makes our algorithm significantly distinct from previously reported algorithms, is that the target covering array $\text{CA}(N; t + 1, k + 1, v)$ is not constructed element by element, but submodule by submodule (each submodule of strength $t$). Nevertheless, our algorithm requires the construction of the non-isomorphic CAs of strength $t$ with $k$ columns, which could had been constructed element by element. However, the cost of constructing the non-isomorphic $\text{CA}(N; t, k, v)$ plus the cost of exploring the juxtapositions of $v$ CAs (produced by permutations of columns and symbols) could be smaller than the cost of constructing $\text{CA}(N; t + 1, k + 1, v)$ element by element.

To end this section, we describe how the proposed algorithm handles the instance $\text{CA}(12; 3, 11, 2)$. In Algorithm 1, we have $k = 10$, $t = 2$, and $S = \{6, 6\}$ because $\text{CAN}(2; 10, 2) = 6$. Since $\text{CA}(6; 2, 10, 2)$ is unique up to isomorphisms, the only tuple $T = (A_0, A_1)$ processed by the function $\text{generate juxtapositions}()$ is the tuple where both $A_0$ and $A_1$ are the minimum $\text{CA}(6; 2, 10, 2)$. Algorithm 2 creates an array $J$ of size $12 \times 11$ whose first 6 rows and 10 columns contain the minimum $\text{CA}(6; 2, 10, 2)$, and the last column is formed by 6 zeros and 6 ones, as shown in Figure 1a. In Algorithm 3 the first three columns of the block $F_1$ are filled with the first three columns of the minimum $\text{CA}(6; 2, 10, 2)$ relabeled with the identity permutation of symbols, as shown in Figure 1a. Since $v = 2$ there are two permutation of symbols: the identity permutation $(0\; 1)$ and the permutation that exchanges 0’s and 1’s $(0\; 1)$; these two symbol permutations are identified by $\varepsilon_0$ and $\varepsilon_1$ respectively.

The first three columns of the entire array $J$ do not form a CA of strength three, for example the tuple $(0, 0, 1)$ does not appear in none of the 12 rows. So, the algorithm tests the next permutation of symbols for the third column of the block $F_1$, which results in the array $J$ of Figure 1b. Again, the first three columns of $J$ do not form a CA of strength three. At this point, the algorithm has tested the two possible
permutation of symbols for the third column of $F_1$ and none relabeling forms a CA of strength three. Thus, the algorithm skips all column permutations whose first three columns are the first three columns of the minimum CA(6;2,10,2).

The algorithm continues and the fourth column of the minimum CA(6;2,10,2) is copied to the third column of $F_1$ using $\varepsilon_0$, as show in Figure 1c. Equally, in this case, the first three columns of $J$ do not form a CA of strength three, and so the next step is to relabel the current third column of $F_1$ using $\varepsilon_1$, which also does not make a CA of strength three. The algorithm continues in this way until all columns 4, . . . , 9 of the minimum CA(6;2,10,2) are copied to the third column of $F_1$ using $\varepsilon_0$ first and then $\varepsilon_1$. After that, the algorithm relabels the current second column of $F_1$ using $\varepsilon_1$ and copies the third column of CA(6;2,10,2) to the third column of $F_1$ obtain the array $J$ of Figure 1d. The algorithm works in this way until all column permutations of CA(6;2,10,2) are assigned to $F_1$ and for each column permutation all possible symbol permutations of its columns are tested. For this instance, the algorithm finds 1 440 arrays ($JE$) that are a CA(12;3,11,2), but all of them produce the same minimum CA(12;3,11,2).

5. Computational results

This section presents the main computational results obtained by executing Algorithm 1. Subsection 5.1 describes the experimentation to determine $CAN(4,13,2)=32$; the implications of this results are the new covering array numbers $CAN(5,14,2)=64$, $CAN(6,15,2)=128$, and $CAN(7,16,2)=256$. Subsection 5.2 shows the steps followed to find the result $CAN(5,8,2)=52$, and Subsection 5.3 gives the steps to obtain the value $CAN(5,9,2)=54$. Subsection 5.4 presents the experimentation done to improve the lower bound
Figure 1: First steps in the processing of the instance CA(12; 3, 11, 2). (a) The first assignment to the first three columns of \( F_1 \). (b) The third column of \( F_1 \) is relabeled using \( \varepsilon_1 \). (c) The fourth column of the minimum \( \text{CA}(6; 2, 10, 2) \) is copied to the third column of \( F_1 \). (d) The second column of \( F_1 \) is relabeled using \( \varepsilon_1 \).

of \( \text{CAN}(6, 9, 2) \) from 96 to 107. Subsection 5.5 shows the computational experimentation to improve the lower bounds of \( \text{CAN}(3, 7, 3), \text{CAN}(3, 9, 3) \), and \( \text{CAN}(4, 7, 3) \). Subsection 5.6 summarizes all the new results found in this work, and Subsection 5.7 uses Algorithm 1 to reproduce some known results as a form of consistency check.

### 5.1. \( \text{CAN}(4, 13, 2) = 32 \)

The current lower bound of \( \text{CAN}(4, 13, 2) \) is 30 [7], and its current upper bound is 32 [19]. In this section we prove that no \( \text{CA}(30; 4, 13, 2) \) and no \( \text{CA}(31; 4, 13, 2) \) exist, and therefore \( \text{CAN}(4, 13, 2) = 32 \).

By Theorem 1 if \( \text{CA}(30; 4, 13, 2) \) exists, then it must be constructed by the vertical juxtaposition of \( \text{CA}(N_0; 3, 12, 2) \) and \( \text{CA}(N_1; 3, 12, 2) \). Now, the only possibility for the values of \( N_0 \) and \( N_1 \) is \( N_0 = N_1 = 15 \) because \( \text{CAN}(3, 12, 2) = 15 \) [7]; so the unique valid multiset in this case is \( \{15, 15\} \). The NonIsoCA algorithm gives two distinct \( \text{CA}(15; 3, 12, 2) \), and by using these CAs, Algorithm 1 did not find a \( \text{CA}(30; 4, 13, 2) \).

Similarly, to determine the existence of \( \text{CA}(31; 4, 13, 2) \), the unique valid multiset is \( \{15, 16\} \). Algorithm 1 tested the juxtapositions of the two non-isomorphic \( \text{CA}(15; 3, 12, 2) \) with the 44291 non-isomorphic \( \text{CA}(16; 3, 12, 2) \) reported by the NonIsoCA algorithm. Also in this case \( \text{CA}(31; 4, 13, 2) \) was not found. Therefore, \( \text{CA}(32; 4, 13, 2) \) is optimal, and \( \text{CAN}(4, 13, 2) = 32 \).
It is possible to run the NonIsoCA algorithm to prove the nonexistence of CA(30; 4, 13, 2) and CA(31; 4, 13, 2) directly in strength four. However, it is more convenient to use the NonIsoCA algorithm to construct the non-isomorphic CAs with strength $t = 3$ and $k = 12$ columns required by Algorithm 1 to search for CA(30; 4, 13, 2) and CA(31; 4, 13, 2). In a machine with processor AMD Opteron™ 6274 at 2.2 GHz the NonIsoCA algorithm takes approximately 1.38 hours to construct the two distinct CA(15; 3, 12, 2), and takes about 937 hours to construct the 44291 distinct CA(16; 3, 12, 2). However, the execution time of Algorithm 1 on the same machine is only 3 seconds for CA(30; 4, 13, 2), and 16 hours for CA(31; 4, 13, 2). Thus, the total time to determine CAN(4, 13, 2) = 32 was approximately 955 hours. On the other hand, we attempted to construct the non-isomorphic CA(31; 4, 13, 2) using the NonIsoCA algorithm, but we aborted the search after 3 months of execution, because based on the partial results, we estimated that the execution would not end soon.

In the process of proving the optimality of CA(32; 4, 13, 2), almost all execution time was consumed in constructing the 44291 non-isomorphic CA(16; 3, 12, 2). We could use Algorithm 1 to construct these CAs; however in this case, Algorithm 1 is not the best option because there are too many CAs with strength $t = 2$ and $k = 11$ columns to be combined to form a CA(16; 3, 12, 2). Since CAN(2, 11, 2) = 7, we can construct a CA(16; 3, 12, 2) by juxtaposing a CA(7; 2, 11, 2) and a CA(9; 2, 11, 2), or by juxtaposing two CA(8; 2, 11, 2). The number of non-isomorphic CA(7; 2, 11, 2) is only 26, but there are 377 177 non-isomorphic CA(8; 2, 11, 2), and 2 148 812 219 distinct CA(9; 2, 11, 2). Thus, in Algorithm 1, the function \textit{generate juxtapositions} () would be called $(26)(2 148 812 219) + 377 177^2$ times.

The covering array CA(32; 4, 13, 2) is also optimal in the number of columns. The value CAN(3, 13, 2) = 16 was proved in [10]; so, the only valid multiset to construct a CA(32; 4, 13, 2) is $\{16, 16\}$. The NonIsoCA algorithm reported 89 distinct CA(16; 3, 13, 2), and using these CAs Algorithm 1 did not find a CA(32; 4, 14, 2), which implies CAK(32; 4, 2) = 13.

The new covering array number CAN(4, 13, 2) = 32 has important consequences. In [18] it was reported a Tower of Covering Arrays (TCA) beginning with CA(8; 2, 11, 2) and ending at CA(256; 7, 16, 2). A TCA is a succession of CAs where the first CA is CA($N$; $t$, $k$, $v$) and the $i$-th CA ($i > 0$) has $N^i$ rows, $k + i$ columns, strength $t + i$, and order $v$. The complete TCA constructed is this:

CA(8; 2, 11, 2), CA(16; 3, 12, 2), CA(32; 4, 13, 2), CA(64; 5, 14, 2), CA(128; 6, 15, 2), CA(256; 7, 16, 2).

The first two CAs of the tower are not optimal because CAN(2, 11, 2) = 7 and CAN(3, 12, 2) = 15. However, from CAN(4, 13, 2) = 32 we have CAN(5, 14, 2) = 64, CAN(6, 15, 2) = 128, and CAN(7, 16, 2) = 256, due to the inequality CAN($t + 1$, $k + 1$, 2) $\geq$ 2 CAN($t$, $k$, 2) [16], which says the optimal CA with $k + 1$ columns and strength $t + 1$ has at least two times the number of rows of the optimal CA with $k$ columns and strength $t$. In a TCA with $v = 2$ every CA, other than the first one, has exactly two times the number of rows of the previous CA, and so if the $i$-th CA is optimal then the $j$-th CAs, $j > i$, are also optimal.

5.2. \textit{CAN}(5, 8, 2) = 52

CAN(5, 8, 2) is the first element of the class CAN($t$, $t + 3$, 2) whose exact value is unknown; its current status is $48 \leq$ CAN(5, 8, 2) $\leq$ 52 [7, 19]. To find CAN(5, 8, 2) we need to check the juxtapositions of the non-isomorphic CA($N_0$; 4, 7, 2) with the non-isomorphic CA($N_1$; 4, 7, 2) for $N_0 + N_1 \in \{48, 49, 50, 51, 52\}$. The first step is to search a CA with 48 rows, if it does not exist, the next step is to search a CA with 49 rows, and so on.
As in the previous subsection, it is possible to run the NonIsoCA algorithm to determine directly in strength \( t = 5 \) if CA(48; 5, 8, 2) exists, but this will take an impractical amount of time. So, the strategy is to use the NonIsoCA algorithm to generate the non-isomorphic CA(24; 4, 7, 2) required in Algorithm 1 to construct CA(48; 5, 8, 2). As shown in Subtable 1a, there is only one non-isomorphic CA(24; 4, 7, 2). Subtable 1b shows that no CA(48; 5, 8, 2) was constructed by Algorithm 1 from the juxtaposition of the unique CA(24; 4, 7, 2) with itself. This result is consistent with the demonstration of the nonexistence of CA(48; 5, 13, 2) done in [4].

Table 1: Computations to find the value of CAN(5, 8, 2). (a) Number of non-isomorphic CA(\(M; 4, 7, 2\)) for \(M = 24, 25, 26, 27, 28\). (b) Number of non-isomorphic CA(\(N; 5, 8, 2\)) constructed by juxtaposing CA(\(N_0; 4, 7, 2\)) and CA(\(N_1; 4, 7, 2\)), where \(N = N_0 + N_1\) and 48 \(\leq N \leq 52\).

| \(M\) | Non-iso | \(N\) | Multisets \(\{N_0, N_1\}\) | Non-iso |
|------|--------|------|-------------------------|--------|
| 24   | 1      | 48   | \{24, 24\}             | 0      |
| 25   | 6      | 49   | \{24, 25\}             | 0      |
| 26   | 228    | 50   | \{24, 26\}, \{25, 25\} | 0      |
| 27   | 13012  | 51   | \{24, 27\}, \{25, 26\} | 0      |
| 28   | 919874 | 52   | \{24, 28\}, \{25, 27\}, \{26, 26\} | 8      |

Now, to search if CA(49; 5, 8, 2) exists, we need to juxtapose the non-isomorphic CA(24; 4, 7, 2) with the non-isomorphic CA(25; 4, 7, 2). There is only one CA(24; 4, 7, 2), and for CA(25; 4, 7, 2) the NonIsoCA algorithm reported 6 distinct CAs. Using these CAs, Algorithm 1 did not find a CA(49; 5, 8, 2). The same strategy is repeated to determine the existence of CA(50; 5, 8, 2), CA(51; 5, 8, 2), and CA(52; 5, 8, 2). From the results in Subtable 1b we have CAN(5, 8, 2) = 52, and there are eight distinct CA(52; 5, 8, 2).

5.3. **CAN(5, 9, 2) = 54**

For CAN(5, 9, 2), the current lower bound is 52 (Subsection 5.2) and the current upper bound is 54 [19]. Then, to determine the exact value of CAN(5, 9, 2) we need to check if there is a CA with 52 or 53 rows. Subtable 2a shows the number of non-isomorphic CA(\(M; 4, 8, 2\)) generated by the NonIsoCA algorithm for \(M = 24, 25, 26, 27, 28\), and 30. These CAs are used to search for the non-isomorphic CA(\(N; 5, 9, 2\)) with \(N = 52, 53, 54\). Subtable 2b shows the valid multisets \(\{N_0, N_1\}\) and the number of non-isomorphic CAs constructed for each \(N \in \{52, 53, 54\}\). From the results we have CAN(5, 9, 2) = 54, and there is only one distinct CA(54; 5, 9, 2), which is shown in Figure 2 (transposed).

![Figure 2](image-url)
Table 2: Computations to find the value of CAN(5, 9, 2). (a) Number of non-isomorphic CA($M; 4, 8, 2$) for $M = 24, \ldots, 30$. (b) Number of non-isomorphic CA($N; 5, 9, 2$) constructed by juxtaposing CA($N_0; 4, 8, 2$) and CA($N_1; 4, 8, 2$), where $N = N_0 + N_1$ and $52 \leq N \leq 54$.

(a) Non-iso CA($M; 4, 8, 2$)  

| $M$  | Non-iso |
|------|---------|
| 24   | 1       |
| 25   | 7       |
| 26   | 195     |
| 27   | 9 045   |
| 28   | 522 573 |
| 29   | 27 826 894 |
| 30   | 1 374 716 212 |

(b) Non-iso CA($N; 5, 9, 2$)

| $N$  | Multisets $\{N_0, N_1\}$ | Non-iso |
|------|--------------------------|---------|
| 52   | $\{24, 28\}, \{25, 27\}, \{26, 26\}$ | 0       |
| 53   | $\{24, 29\}, \{25, 28\}, \{26, 27\}$ | 0       |
| 54   | $\{24, 30\}, \{25, 29\}, \{26, 28\}, \{27, 27\}$ | 1       |

The ExtendNonIsoCA algorithm executed with the unique CA($54; 5, 9, 2$) produced zero CA($54; 5, 10, 2$); then, CAK($54; 5, 2$) = 9. This result improves from 52 to 55 the lower bounds of CAN($5, 10, 2$), CAN($5, 11, 2$), CAN($5, 12, 2$), and CAN($5, 13, 2$), taken from [7].

5.4. Improving the lower bound of CAN($6, 9, 2$)

The next CAN of the class CAN($t, t + 3, 2$) to be determined is CAN($6, 9, 2$). Its current status is 96 \leq CAN($6, 9, 2$) \leq 108. From CAN($5, 8, 2$) = 52 (Subsection 5.2) and from the inequality CAN($t + 1, k + 1, 2$) \geq 2 CAN($t, k, 2$) we have CAN($6, 9, 2$) \geq 104. Therefore, the new lower bound of CAN($6, 9, 2$) is 104, but we can improve further this lower bound by using the algorithm developed in this work.

Firstly, the juxtapositions of the 8 non-isomorphic CA($52; 5, 8, 2$) found in Subsection 5.2 with themselves do not produce a CA($104; 6, 9, 2$); therefore CAN($6, 9, 2$) \geq 105.

To determine the existence of CA($105; 6, 9, 2$) we need to test the juxtapositions of the non-isomorphic CA($52; 5, 8, 2$) with the non-isomorphic CA($53; 5, 8, 2$). But to obtain the non-isomorphic CA($53; 5, 8, 2$) we need to juxtapose CA($N_0; 4, 7, 2$) and CA($N_1; 4, 7, 2$) where $N_0 + N_1 = 53$. Previously in Subsection 5.2, the non-isomorphic CA($M; 4, 7, 2$) for $M = 24, \ldots, 28$ were generated, so we only need to generate the non-isomorphic CA($29; 4, 7, 2$) to have all CAs with $t = 4$ and $k = 7$ required to construct CA($53; 5, 8, 2$). The NonIsoCA algorithm reported 58 488 647 distinct CA($29; 4, 7, 2$), as shown in Subtable 3a. Subtable 3b shows the multisets for $N = 53$ and the number of non-isomorphic CA($53; 5, 8, 2$) constructed by Algorithm 1; in this case there are 213 distinct CA($53; 5, 8, 2$). Subtable 3c shows the result of juxtaposing the non-isomorphic CA($52; 5, 8, 2$) with the non-isomorphic CA($53; 5, 8, 2$) to try to construct CA($105; 6, 9, 2$). No CA($105; 6, 9, 2$) was generated, then CAN($6, 9, 2$) \geq 106.

Note that we are using the non-isomorphic CAs generated by Algorithm 1 in another execution of it, because from the non-isomorphic CAs with $t = 4$ and $k = 7$ are constructed the non-isomorphic CAs with $t = 5$ and $k = 8$, and these last CAs are used to search for the non-isomorphic CAs with $t = 6$ and $k = 9$.

Now, to determine if CA($106; 6, 9, 2$) exists, we first compute the valid multisets $\{L_0, L_1\}$ such that the juxtaposition of CA($L_0; 5, 8, 2$) and CA($L_1; 5, 8, 2$) might produce CA($106; 6, 9, 2$). In this case there are two possibilities: $\{52, 54\}$ and $\{53, 53\}$. The non-isomorphic CA($52; 5, 8, 2$) and CA($53; 5, 8, 2$) have been constructed previously, but it remains to construct the distinct CA($54; 5, 8, 2$). To do this, we juxtapose the non-isomorphic CA($N_0; 4, 7, 2$) with the non-isomorphic CA($N_1; 4, 7, 2$) where $N_0 + N_1 = 54$. Subtable 3a shows that there are 3 177 398 378 distinct CA($30; 4, 7, 2$). Subtable 3b shows the results of juxtaposing...
Table 3: Computations to improve the lower bound of CAN(6, 9, 2). (a) Number of non-isomorphic CA\(M; 4, 7, 2\) for \(M = 29, 30\). (b) Number of non-isomorphic \(CA(N; 5, 8, 2)\) constructed by juxtaposing \(CA(N_0; 4, 7, 2)\) and \(CA(N_1; 4, 7, 2)\), where \(N = N_0 + N_1\) and 53 \(\leq N \leq 54\). (c) Number of non-isomorphic \(CA(L; 6, 9, 2)\) constructed by juxtaposing \(CA(L_0; 5, 8, 2)\) and \(CA(L_1; 5, 8, 2)\), with \(L = L_0 + L_1\) and 104 \(\leq L \leq 106\).

| \(M\) | Non-iso | \(N\) | Multisets \(\{N_0, N_1\}\) | Non-iso |
|-------|---------|-------|-----------------|---------|
| 29    | 58 488 467 | 53    | \{24, 29\}, \{25, 28\}, \{26, 27\} | 213     |
| 30    | 3 177 398 378 | 54    | \{24, 30\}, \{25, 29\}, \{26, 28\}, \{27, 27\} | 20 450  |

(c) Non-iso \(CA(L; 6, 9, 2)\)

| \(L\) | Multisets \(\{L_0, L_1\}\) | Non-iso |
|-------|-----------------|---------|
| 104   | \{52, 52\}    | 0       |
| 105   | \{52, 53\}    | 0       |
| 106   | \{52, 54\}, \{53, 53\} | 0       |

CA\((N_0; 4, 7, 2)\) and \(CA(N_1; 4, 7, 2)\) where \(N_0 + N_1 = 54\); in total there are 20 450 distinct \(CA(54; 5, 8, 2)\). Subtable 3c contains the result of juxtaposing the distinct \(CA(52; 5, 8, 2)\) with the distinct \(CA(54; 5, 8, 2)\), and the distinct \(CA(53; 5, 8, 2)\) with themselves. No \(CA(106; 6, 9, 2)\) was generated, thus \(CAN(6, 9, 2) \geq 107\).

It was not possible to determine the existence of \(CA(107; 6, 9, 2)\) due to the huge computational time required to construct the non-isomorphic \(CA(55; 5, 8, 2)\). However, the result \(CAN(6, 9, 2) \geq 107\) improves the lower bounds of \(CAN(t, t + 3, 2)\) for \(7 \leq t \leq 11\); their new values are:

- \(214 \leq CAN(7, 10, 2) \leq 222\)
- \(428 \leq CAN(8, 11, 2) \leq 496\)
- \(856 \leq CAN(9, 12, 2) \leq 992\)
- \(1712 \leq CAN(10, 13, 2) \leq 2016\)
- \(3424 \leq CAN(11, 14, 2) \leq 4032\)

Upper bounds were taken from [7] for \(t = 7\), and from [17] for \(t = 8, 9, 10, 11\).

5.5. Results for \(v = 3\)

This section presents the computational results obtained for CAs with order \(v = 3\). The results are given in a list format. Lower and upper bounds were taken respectively from [7] and [5]:

- **There is a unique CA\((33; 3, 6, 3)\).** This CA is known to be optimal [3], but we prove its uniqueness. Since \(CAN(2, 5, 3) = 11\), the only valid multiset to construct \(CA(33; 3, 6, 3)\) is \(\{11, 11, 11\}\). The NonIsoCA algorithm reported 3 non-isomorphic \(CA(11; 2, 5, 3)\), and using these CAs, Algorithm 1 constructed only one \(CA(33; 3, 6, 3)\), which is shown next (transposed):
• Nonexistence of CA(99; 4, 7, 3). The current status of CAN(4, 7, 3) is $99 \leq \text{CAN}(4, 7, 3) \leq 123$. Using the unique CA(33; 3, 6, 3), Algorithm 1 determined the nonexistence of CA(99; 4, 7, 3). Therefore, $100 \leq \text{CAN}(4, 7, 3) \leq 123$.

• Nonexistence of CA(36; 3, 7, 3). Currently $36 \leq \text{CAN}(3, 7, 3) \leq 39$. Since CAN(2, 6, 3) = 12, the only way to form a CA(36; 3, 7, 3) is by juxtaposing three covering arrays CA(12; 2, 6, 3). The Non-IsoCA algorithm produced 13 non-isomorphic CA(12; 2, 6, 3), from which Algorithm 1 did not find a CA(36; 3, 7, 3). Thus, $37 \leq \text{CAN}(3, 7, 3) \leq 39$.

• Nonexistence of CA(39; 3, 9, 3). The current lower bound of CAN(3, 9, 3) is 39 and its current upper bound is 45. Because CAN(2, 8, 3) = 13, the only possibility to form a CA(39; 3, 9, 3) is by juxtaposing three CA(13; 2, 8, 3). The number of non-isomorphic CA(13; 2, 8, 3) constructed by the NonIsoCA algorithm is five. Using these CAs Algorithm 1 searched for CA(39; 3, 9, 3) but no such CA was found. Therefore, $40 \leq \text{CAN}(3, 9, 3) \leq 45$.

5.6. Summary of the new results

Now, we present a summary of the results achieved by the new algorithm to construct non-isomorphic CAs. Among the results are new CANs, number of isomorphism classes, and improvements in the lower bounds of some CANs. The new results are listed in Table 4.

Table 4: Summary of the results. (a) The new covering array numbers. (b) The cases where the number of distinct CAs was found. (c) The improved lower bounds.

| (a) CAN(t, k, v) = N | (b) CA | Non-iso | (c) CAN(t, k, v) ≤ LB |
|---------------------|--------|--------|-----------------------|
| CAN(4, 13, 2) = 32  | CAN(52; 5, 8, 2) | 8      | CAN(5, 10, 2) ≥ 55   |
| CAN(5, 14, 2) = 64  | CAN(53; 5, 8, 2) | 213    | CAN(5, 11, 2) ≥ 55   |
| CAN(6, 15, 2) = 128 | CAN(54; 5, 8, 2) | 20450  | CAN(5, 12, 2) ≥ 55   |
| CAN(7, 16, 2) = 256 | CAN(54; 5, 9, 2) | 1      | CAN(5, 13, 2) ≥ 55   |
| CAN(5, 8, 2) = 52   | CAN(33; 3, 6, 3) | 1      | CAN(6, 9, 2) ≥ 107   |
| CAN(5, 9, 2) = 54   |          |        | CAN(7, 10, 2) ≥ 214  |
|                     |          |        | CAN(8, 11, 2) ≥ 428  |
|                     |          |        | CAN(9, 12, 2) ≥ 856  |
|                     |          |        | CAN(10, 13, 2) ≥ 1712|
|                     |          |        | CAN(11, 14, 2) ≥ 3424|
|                     |          |        | CAN(4, 7, 3) ≥ 100   |
|                     |          |        | CAN(3, 7, 3) ≥ 37    |
|                     |          |        | CAN(3, 9, 3) ≥ 40    |
5.7. Construction of known results for \( v = 2 \) and \( t = 3, 4 \)

To validate that Algorithm 1 generates correct results, we constructed the non-isomorphic CAs for some known optimal cases of order \( v = 2 \) and strengths \( t = 3, 4 \). The idea is to compare the results of Algorithm 1 with the results of the NonIsoCA algorithm of [10] to see if the results of both algorithms match perfectly. For both strengths three and four, we constructed the non-isomorphic CAs up to \( k = 12 \) columns. The results obtained are shown in Table 5; CA 1 and CA 2 are the two CAs juxtaposed to construct the CA in the first column; the columns with header \#Non-iso indicates the number of non-isomorphic CAs that exist for the CA in the preceding column. We can verify that the results in the first two columns of the table match the results reported in [10].

Table 5: Construction of non-isomorphic CAs to reproduce some of the results of [10].

| CA constructed | \#Non-iso | CA 1 | \#Non-iso | CA 2 | \#Non-iso |
|----------------|-----------|------|-----------|------|-----------|
| CA(8; 3, 3, 2) | 1         | CA(4; 2, 2, 2) | 1         | CA(4; 2, 2, 2) | 1         |
| CA(8; 3, 4, 2) | 1         | CA(4; 2, 3, 2) | 1         | CA(4; 2, 3, 2) | 1         |
| CA(10; 3, 5, 2)| 1         | CA(5; 2, 4, 2) | 1         | CA(5; 2, 4, 2) | 1         |
| CA(12; 3, 6, 2)| 9         | CA(6; 2, 5, 2) | 7         | CA(6; 2, 5, 2) | 7         |
| CA(12; 3, 7, 2)| 2         | CA(6; 2, 6, 2) | 4         | CA(6; 2, 6, 2) | 4         |
| CA(12; 3, 8, 2)| 2         | CA(6; 2, 7, 2) | 3         | CA(6; 2, 7, 2) | 3         |
| CA(12; 3, 9, 2)| 1         | CA(6; 2, 8, 2) | 1         | CA(6; 2, 8, 2) | 1         |
| CA(12; 3, 10, 2)| 1       | CA(6; 2, 9, 2) | 1         | CA(6; 2, 9, 2) | 1         |
| CA(12; 3, 11, 2)| 1       | CA(6; 2, 10, 2)| 1        | CA(6; 2, 10, 2)| 1         |
| CA(15; 3, 12, 2)| 2       | CA(7; 2, 11, 2)| 26        | CA(8; 2, 11, 2)| 377177    |
| CA(16; 4, 4, 2) | 1         | CA(8; 3, 3, 2) | 1         | CA(8; 3, 3, 2) | 1         |
| CA(16; 4, 5, 2) | 1         | CA(8; 3, 4, 2) | 1         | CA(8; 3, 4, 2) | 1         |
| CA(21; 4, 6, 2) | 1         | CA(10; 3, 5, 2)| 1         | CA(11; 3, 5, 2) | 4         |
| CA(24; 4, 7, 2) | 1         | CA(12; 3, 6, 2)| 9         | CA(12; 3, 6, 2) | 9         |
| CA(24; 4, 8, 2) | 1         | CA(12; 3, 7, 2)| 2         | CA(12; 3, 7, 2) | 2         |
| CA(24; 4, 9, 2) | 1         | CA(12; 3, 8, 2)| 2         | CA(12; 3, 8, 2) | 2         |
| CA(24; 4, 10, 2)| 1         | CA(12; 3, 9, 2)| 1         | CA(12; 3, 9, 2) | 1         |
| CA(24; 4, 11, 2)| 1         | CA(12; 3, 10, 2)| 1        | CA(12; 3, 10, 2)| 1         |
| CA(24; 4, 12, 2)| 1         | CA(12; 3, 11, 2)| 1        | CA(12; 3, 11, 2)| 1         |

6. Conclusions

If a CA(\(N; t + 1, k + 1, v\)) exists, then it can be constructed from the juxtaposition of \( v \) covering arrays CA(\(N_0; t, k, v\)), CA(\(N_1; t, k, v\)), . . . , CA(\(N_{v-1}; t, k, v\)) \(N = \sum_{i=0}^{v-1} N_i\) plus a column formed by concatenating \(N_i\) elements equal to \( i \) for \( 0 \leq i \leq v - 1 \). Because of this, we developed an algorithm to determine the existence or nonexistence of CA(\(N; t + 1, k + 1, v\)) by testing all possible juxtapositions of \( v \) CAs with strength \( t \) and \( k \) columns. If none juxtaposition generates a CA(\(N; t + 1, k + 1, v\)), then a CA with these parameters does not exist. If the existence of CA(\(N; t + 1, k + 1, v\)) is known, and we prove the nonexistence of CA(\(N - 1; t + 1, k + 1, v\)), then we have CAN(t + 1, k + 1, v) = \(N\).

Using this algorithm, we found the following covering array numbers: CAN(13, 4, 2) = 32, CAN(5, 8, 2) = 52, and CAN(5, 9, 2) = 54. To the best of our knowledge, these CANs had not been determined before. The optimality of CA(32; 4, 13, 2) implies the optimality of CA(64; 5, 14, 2), CA(128; 6, 15, 2), and
CA(256; 7, 16, 2), due to some properties of CAs. Thus, the implications of CAN(4, 13, 2) = 32 are very important, since without this result, for example, we would have to prove the nonexistence of CA(255; 7, 16, 2) to conclude the optimality of CA(256; 7, 16, 2), but the instance CA(255; 7, 16, 2) is too large for an exact algorithm. Another important result is the improvement of the lower bound of CAN(6, 9, 2) from 96 to 107, which in turns improves the lower bound of CAN(t, t + 3, 2) for t = 7, 8, 9, 10, 11. For v = 3, the results obtained were the uniqueness of CA(33; 3, 6, 3), and the nonexistence of CA(99; 4, 7, 3), CA(36; 3, 7, 3), and CA(39; 3, 9, 3).

Our algorithm required a lot of computational time to determine these new covering array numbers; for example CAN(13, 4, 2) = 32 took over a month. However, the instances processed in this work are of considerable size to be handled by an exact algorithm. For some instances, our algorithm is faster than previous methods because it searches for a CA(N; t + 1, k + 1, v) with a certain structure; this CA is formed by v blocks, where each block is not an arbitrary array but a CA with strength t and k columns; this allows our algorithm to handle larger instances.

[1] Bracho-Rios, J., Torres-Jimenez, J., Rodriguez-Tello, E., 2009. A new backtracking algorithm for constructing binary covering arrays of variable strength. In: Aguirre, A. H., Borja, R. M., García, C. A. R. (Eds.), MICAI 2009: Advances in Artificial Intelligence. Vol. 5845 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, pp. 397–407.

[2] Bush, K. A., 1952. Orthogonal arrays of index unity. Ann. Math. Stat. 23 (3), 426–434.

[3] Chateauneuf, M. A., Colbourn, C. J., Kreher, D. L., 1999. Covering arrays of strength three. Des. Codes Cryptogr. 16 (3), 235–242.

[4] Choi, S., Kim, H. K., Oh, D. Y., 2012. Structures and lower bounds for binary covering arrays. Discrete Math. 312 (19), 2958–2968.

[5] Colbourn, C. J., 2018. Covering array tables for t = 2, 3, 4, 5, 6, last time accessed on sep 30, 2018.

URL http://www.public.asu.edu/~ccolbou/src/tabby/catable.html

[6] Colbourn, C. J., Dinitz, J. H., 2006. Handbook of Combinatorial Designs, 2nd Edition. Chapman and Hall/CRC.

[7] Colbourn, C. J., Kéri, G., Soriani, P. P. R., Schlage-Puchta, J. C., Jun. 2010. Covering and radius-covering arrays: Constructions and classification. Discrete Appl. Math. 158 (11), 1158–1180.

[8] Hedayat, A. S., Sloane, N. J. A., Stufken, J., 1999. Orthogonal Arrays. Springer-Verlag New York.

[9] Hnich, B., Prestwich, S. D., Selensky, E., Smith, B. M., Jul. 2006. Constraint models for the covering test problem. Constraints 11 (2–3), 199–219.

[10] Izquierdo-Marcuez, I., Torres-Jimenez, J., 2018. New optimal covering arrays using an orderly algorithm. Discrete Mathematics, Algorithms and Applications 10 (01), 1850011.

[11] Johnson, K. A., Entringer, R., 1989. Largest induced subgraphs of the n-cube that contain no 4-cycles. J. Combin. Theory Ser. B 46 (3), 346–355.

[12] Katona, G. O. H., 1973. Two applications (for search theory and truth functions) of sperner type theorems. Period. Math. Hungar. 3 (1–2), 19–26.

[13] Kitsos, P., Simos, D. E., Torres-Jimenez, J., Voyiatzis, A. G., Nov 2015. Exciting fpga cryptographic trojans using combinatorial testing. In: Software Reliability Engineering (ISSRE), 2015 IEEE 26th International Symposium on. pp. 69–76.

[14] Kleitman, D. J., Spencer, J., 1973. Families of k-independent sets. Discrete Math. 6 (3), 255–262.

[15] Kokkala, J. I., 2017. Computational methods for classification of codes. Ph.D. thesis, Aalto University.

[16] Lawrence, J., Kack, R. N., Lei, Y., Kuhn, D. R., Forbes, M., 2011. A survey of binary covering arrays. Electron. J. Combin. 18 (1), P84.
[17] Torres-Jimenez, J., Izquierdo-Marquez, I., Gonzalez-Gomez, A., Avila-George, H., 2015. A branch & bound algorithm to derive a direct construction for binary covering arrays. In: Sidorov, G., Galicia-Haro, N. S. (Eds.), Advances in Artificial Intelligence and Soft Computing. Springer International Publishing, Cham, pp. 158–177.

[18] Torres-Jimenez, J., Izquierdo-Marquez, I., Kacker, R. N., Kuhn, D. R., 2015. Tower of covering arrays. Discrete Appl. Math. 190–191, 141–146.

[19] Torres-Jimenez, J., Rodriguez-Tello, E., 2012. New bounds for binary covering arrays using simulated annealing. Inform. Sci. 185 (1), 137–152.

[20] Yan, J., Zhang, J., 2006. Backtracking algorithms and search heuristics to generate test suites for combinatorial testing. In: Proceedings of the 30th Annual International Computer Software and Applications Conference - Volume 01. COMPSAC ’06. IEEE Computer Society, Washington, DC, USA, pp. 385–394.