BULLETIN OF THE ICA
Volume **102** (2024), 93–115

Optimal ordered covering arrays via an exact algorithm

Irene Hiess, Ludwig Kampel* and Dimitris E. Simos

Abstract. Ordered covering arrays (orCAs) are combinatorial objects that recently raised interest as they can be used for the generation of covering codes in Niederreiter-Rosenbloom-Tsfasman (NRT) spaces. We present an exact algorithm for the generation of orCAs and use this algorithm to determine sizes of some optimal orCAs. With the calculated orCAs and a theorem from Castoldi et al., *J. Combin. Des.*, 2023, we can phrase inequalities giving upper bounds on the size of covering codes in NRT spaces. We discuss these in the context of bounds that can be derived based on results existing in the literature.

1 Introduction

Orthogonal arrays (OAs) are central combinatorial objects that have been much studied due to their applications and versatile connections with other combinatorial designs, see for example [10] for a thorough overview of OAs and [7] for their connections. An OA can be described as an array with the property that all combinations of t values, *i.e.*, all t-tuples, occur *exactly* λ times in *every* selection of t columns, for parameters λ and t. A generalization of OAs are so called *ordered orthogonal arrays* (OOAs), which have been proposed independently in [17] and [20]. OOAs generalize OAs by weakening their defining property: instead of considering every selection of t columns, for OOAs this defining property only has to hold on *some* selections of t columns, *i.e.*, column selections fulfilling a certain property—formally defined later in this paper. Covering arrays (CAs) are another generalization of OAs, where a different part of the defining property of

*Corresponding author: lkampel@sba-research.org
Key words and phrases: covering arrays, covering codes, SAT solving
AMS (MOS) Subject Classifications: 05B15, 05B40, 68W30

Received: 14 March 2023
Accepted: 29 March 2023

OAs is weakened: instead of requiring that all t-tuples occur exactly λ times in every selection of t columns, it is sufficient if they occur *at least* λ times in every such column selection. CAs have been subject to many studies due to their applications, most prominently in software testing [16], where they serve as a basis for interaction testing. Their aspects with regards to combinatorial designs have also been the focus of several works, see for example [6] for a survey. The focus of this work lies on ordered covering arrays (orCAs), which can be understood a generalization of all three structures, OAs, OOAs and CAs, since they combine the generalizations of OOAs and CAs: in an orCA, every t-tuple has to occur *at least* λ times on *some* selections of t columns, where the considered column selections are the same as for OOAs. In [15], orCAs have been first considered, where several constructions were investigated and their application to quasi-Monte Carlo numerical integration was discussed. In [3] and [4] orCAs have been further explored and applied to generate covering codes in Niederreiter-Rosenbloom-Tsfasman (NRT) spaces. Using bounds on orCA sizes it was possible to derive new bounds on the size of such covering codes. Additionally, constructions for orCAs are given, as well as bounds on the size of orCAs.

In this work, we perform computational enumeration to address the problem of determining the optimal size of an orCA. With an exact algorithm we are able to find minimal orCAs for several instances. From these, we derive some bounds on covering codes in NRT spaces using a known construction. These bounds are discussed briefly and put into context to existing ones. This paper is organized as follows: Section 2 presents some preliminaries and in Section 3 related work is discussed. In Section 4 our exact algorithm is described, which is used to compute sizes of optimal orCAs presented in Section 5. There we also give the derived bounds for covering codes in NRT spaces. Section 6 summarizes this paper and outlines directions of future work.

2 Preliminaries

Below we review preliminaries of orCAs and related notions, necessary in the following.

CAs are a well-known combinatorial structure that is closely related to orCAs. We define CAs analogously to [7]:

Definition 2.1. For positive integers λ, t, k and v, where $2 \leq t \leq k$, a *covering array* $\mathsf{CA}_\lambda(N; t, k, v)$ is an $N \times k$ array with entries from an alphabet of size v, such that in every set of t columns, every t-tuple over the alphabet occurs at least λ times as a row. The parameter t is called the *strength* of the CA. When $\lambda = 1$, the subscript can be omitted from the notation.

The defining condition of CAs can also be expressed using the notion of t-way interactions.

Definition 2.2. A *t-way interaction* for a $\mathsf{CA}_\lambda(N; t, k, v)$ is a selection of t columns, together with t values of the alphabet underlying the considered CA. A t-way interaction τ is formally written as $\tau = \{(p_1, v_1), \ldots, (p_t, v_t)\}$ where for all $1 \leq i \leq t$ the p_i are pairwise different column indices and the v_i raise from the underlying alphabet.

We say that a t-way interaction is *covered* by a CA A if A has a row $(r_1, \ldots, r_k)$ with $r_{p_i} = v_i$ for $1 \leq i \leq t$. This also motivates the name *covering array*, as a $\mathsf{CA}(N; t, k, v)$ covers all t-way interactions. A covering array is called *optimal* if its number of rows is minimal when the remaining parameters are fixed. The covering array number (CAN) denotes the optimal size of a CA.

Definition 2.3. The *covering array number* $\mathsf{CAN}_\lambda(t, k, v)$ is the minimum integer N for which a $\mathsf{CA}_\lambda(N; t, k, v)$ exists.

Variable strength covering arrays (*VCAs*) are a generalization of CAs, where tuples on a specified set of column selections need to be covered, where the size of the column selections can be varied [23]. *Ordered covering arrays* can be understood as a special case of VCAs, where only a specific set of column selections can be considered. They are defined in [4] using the notion of Niederreiter-Rosenbloom-Tsfasman posets (NRT-posets). For the sake of simplicity we adopt the ordered CA definition from [15], however, we denote ordered CAs as orCA instead of OCA because OCA is also used in the literature to denote optimal covering arrays, see [13], for example.

Definition 2.4. Given positive integers m and s, and an array with ms columns, labelled $\{(i, j) : 1 \leq i \leq m, 1 \leq j \leq s\}$, a set T of columns is *right-justified* if and only if for every $j < s$

$$(i, j) \in T \implies (i, j + 1) \in T.$$

When we view an array with ms columns as a juxtaposition of m arrays, each having s columns, this motivates the above notation, since a set of right-justified columns consists of *all columns to the right* of some column for each sub-array with s columns. In the following we refer to these sub-arrays with s columns as *s-blocks.* Figure 2.1 gives an illustration of an array consisting of three such s-blocks and in equation (1) we list the corresponding right-justified sets of three columns.

Definition 2.5. For positive integers λ, t, m, s and v, where $s \leq t \leq ms$, an *ordered covering array* $\mathsf{orCA}_\lambda(N; t, m, s, v)$ is an $N \times ms$ array with entries from an alphabet of size v, such that in every right-justified set of t columns, every t-tuple occurs at least λ times as a row. The parameter t is called the *strength* of the orCA. When $\lambda = 1$, the subscript can be omitted from the notation.

In this work we only consider orCAs with $\lambda = 1$. The ordered covering array number (orCAN) is defined analogously to CAN.

Definition 2.6. The *ordered covering array number* $\mathsf{orCAN}_\lambda(t, m, s, v)$ is the minimum integer N for which an $\mathsf{orCA}_\lambda(N; t, m, s, v)$ exists.

As also pointed out in [3], for the case $s = 1$ an $\mathsf{orCA}(N; t, m, 1, v)$ is a $\mathsf{CA}(N; t, m, v)$. The difference of orCAs to CAs is that not all t-way interactions need to be covered, but only those occurring on right-justified sets of columns. An example of an $\mathsf{orCA}(10; 3, 3, 2, 2)$, that is not a CA, is given in Figure 2.1. While the array is not a CA of strength 3 (for example the tuple $(0, 0, 1)$ is not covered in the sub-array consisting of the columns labeled with $(1, 1)$, $(2, 1)$ and $(3, 1)$), it is an orCA of strength 3, because in every right-justified set of three columns, every 3-tuple over a binary alphabet occurs at least once, *i.e.*, all 3-way interactions in the following column sets are covered:

$$\left.\begin{array}{l}\{(1,2),(2,2),(3,2)\},\{(1,1),(1,2),(2,2)\},\{(1,1),(1,2),(3,2)\},\\ \{(1,2),(2,1),(2,2)\},\{(2,1),(2,2),(3,2)\},\{(1,2),(3,1),(3,2)\},\\ \{(2,2),(3,1),(3,2)\}\end{array}\right\} \quad (1)$$

Remark 2.7. Clearly, as for $s > t$ the left-most columns of an s-block do not appear in any of the right-justified column selections, we are only interested in orCAs with $s \leq t$. A basic property of orCAs is given in [3, Corollary 1], that says an $\mathsf{orCA}(N; t, m, t, v)$ exists if and only if an $\mathsf{orCA}(N; t, m, t-1, v)$ exists, which implies $\mathsf{orCAN}(N; t, m, t, v) = \mathsf{orCAN}(N; t, m, t-1, v)$. Thus we can further focus on orCAs with $s < t$.

$(1,1)$	$(1,2)$	$(2,1)$	$(2,2)$	$(3,1)$	$(3,2)$
0	0	0	0	0	0
0	0	1	0	1	0
1	0	0	0	1	1
1	0	1	1	0	0
0	0	0	1	0	1
0	1	0	0	0	0
1	1	1	0	0	1
1	1	0	1	1	0
0	1	1	1	1	1
1	1	0	1	1	1

Figure 2.1: An $\mathsf{orCA}(10;3,3,2,2)$: An orCA of strength 3 with 10 rows, 3 s-blocks with $s=2$ and a binary alphabet.

For the special case $t=s$, $s\geq 3$, v a prime power and $2\leq m\leq v+1$ the orCANs were determined in [4] as $\mathsf{orCAN}(s,m,s,v)=v^s$.

3 Related work

Ordered covering arrays were first introduced in Krikorian's Master thesis [15] as special case of variable strength covering arrays (VCAs), where several combinatorial constructions of orCAs were described. In [3] a connection between orCAs and covering codes in NRT spaces was established. In addition to theoretical results and constructions for orCAs and covering codes in NRT spaces, ways to derive upper bounds on covering codes in NRT spaces from orCANs were presented. Those results were extended in [4]. In particular, additional constructions for orCAs were given, together with a table of upper bounds on covering codes in NRT spaces that can be derived from the presented theoretical results for orCANs.

Ordered orthogonal arrays were first considered in [17] and [20], where their equivalence to (t,m,s)-nets was shown. Martin and Stinson [18] generalized the Rao bound for OAs so that it applies also for OOAs, which lead to improvements of the strongest bounds for several parameter settings. In [22] Panario *et al.*, use linear feedback shift register sequences defined by polynomials for the generation of OOAs. An interesting result is that for some parameter settings the primitivity of the polynomial can hinder the generation of an OOA achieving best coverage when using their construction.

While not much literature exists on orCAs, CAs are related structures that have been much studied. Due to their importance in *combinatorial testing* (a branch of software testing) many algorithms and tools for CA generation exist. Below we briefly describe some related work on exact algorithms for CA generation. One of the first exact approaches for the generation of CAs was presented in [11]. Using a translation of the CA generation problem for given parameters N, t, k, v to a CSP formulation, the existence of a $\mathsf{CA}(N; t, k, v)$ was determined using a complete CSP solver. A negative result (no solution satisfying the CSP formulation exists) allowed to derive a lower bound on a covering array number. Further, a SAT formulation for CA generation was described, however, only incomplete local search SAT solvers were applied to the SAT formulation. SAT solving has also been applied to generate closely related structures, such as CAs with constraints [21]. A study on bounds on CANs and bounds on the size of another generalization of CAs (namely radius-covering arrays) can be found in [8]. Theoretical bounds and computational results are presented therein and used to derive new lower and upper bounds on some CANs. Additionally, for several instances a classification of CAs and radius-covering arrays is given. In [14] a concept of *balance* for CAs was introduced, together with a classification algorithm that works as follows: Starting with an empty array and a fixed number of rows, using depth-first search the array is extended step-by-step with additional columns. Using backtracking the complete search space is explored and all CAs of the given size are visited during the search. After every extension step a SAT or pseudo-boolean solver is used to calculate the columns feasible for the next extension. Symmetry breaking is included in the generated SAT formulas for search space reduction. An additional feasibility check is performed to break remaining symmetries. Taking this algorithm as a starting point, the algorithm presented in the next section makes also use of these techniques, adapted for orCA generation.

4 Algorithm

In this section we present our exact algorithm orCA-EXT, which is a modified version of the classification algorithm for CAs described in [14], adapted for finding an $\mathsf{orCA}(N; t, m, s, v)$ for given parameters N, t, m and s and a binary alphabet ($v = 2$). In contrast to the work in [14], we are not interested in a classification of all orCAs of a given size but only in determining orCAN bounds. Thus our algorithm terminates after finding some orCA of the desired size instead of enumerating all solutions. We first outline the algorithm, detail thereafter how SAT solving is used and how the feasibility check for symmetry breaking is employed by the algorithm.

A pseudo-code of orCA-EXT can be found in Algorithm 4.1. This algorithm recursively branches over all possible columns (vectors of length N) that can be added to a given array, and might be part of a final solution, *i.e.*, the orCA to be constructed (lines 5 and 6). The columns that might be part of a solution are calculated in line 4 and stored in a variable *columns*. This calculation of admissible columns uses a two-step approach consisting of a SAT solver and an algorithmic verification of each column. More details on this are explained in Section 4.1. If an orCA of the desired size is found or the given array is already an $\mathsf{orCA}(N; t, m, s, 2)$, the orCA is immediately reported and the algorithm terminates (lines 1–3 and 7–9). If the whole search tree is enumerated and no solution was found, the algorithm returns that no solution exists (line 11). When starting with an empty array, *i.e.*, orCA-EXT$(\emptyset, N, t, m, s)$, the output of the algorithm is either an $\mathsf{orCA}(N; t, m, s, 2)$ or False if no $\mathsf{orCA}(N; t, m, s, 2)$ exists.

Algorithm 4.1 orCA-EXT(A, N, t, m, s)

```
1: if $A$ is an orCA with $ms$ columns then
2:     return $A$                                  ▷ Report found orCA and terminate
3: end if
4: columns ← admissible columns of $A$
5: for all $col \in columns$ do
6:     $result \leftarrow$ orCA-EXT$([A, col], N, t, m, s)$   ▷ Recursively extend $A$
7:     if $result$ is an orCA then
8:         return $result$
9:     end if
10: end for
11: return False
```

The difference between CAs and orCAs is that not all t-way interactions have to be covered in an orCA. The reduced coverage requirement leads to some columns being highly constrained as they occur in many right-justified sets of columns, while other columns occur only in a small number of t-way interactions and therefore allow for more freedom when choosing a column. For a column with index (i, j), the greater j is, the higher is the number of right-justified column sets containing this column. To reduce the effort of searching for an orCA, it is advantageous to have fewer branches at higher levels of our search tree. To achieve this, we generate the columns of the orCA such that highly constrained columns, *i.e.*, those occurring in many t-way interactions, are generated first. More specifically, for an $\mathsf{orCA}(N; t, m, s, 2)$ we define a bijective function $\varphi_{m,s} : \{1, \dots, m\} \times \{1, \dots, s\} \to \{1, \dots, ms\}$ mapping a column label (i, j) to a column index as follows: $\varphi_{m,s}((i, j)) := m(s-j)+i$. Our algorithm generates columns in the order given by $\varphi_{m,s}$, for example for an $\mathsf{orCA}(N; 2, 2, 2, 2)$ the column $(1, 2)$ is processed first because $\varphi_{2,2}((1, 2)) = 1$. We denote with $\varphi^{-1}_{m,s}$ the inverse function of $\varphi_{m,s}$, where $\varphi^{-1}_{m,s}(n) = (((n-1) \mod m)+1, s-\lfloor \frac{n-1}{m} \rfloor)$. In the remainder of this section we depict all (partial) orCAs with columns permuted according to $\varphi_{m,s}$, *i.e.*, they appear in the ordering used by our algorithm.

Example 4.1. Assume we want to generate an $\mathsf{orCA}(4; 2, 2, 2, 2)$. Starting with an empty array, a first set of admissible columns is generated. These columns are $(0, 0, 0, 1)^T$, $(0, 0, 1, 1)^T$ and $(0, 0, 0, 0)^T$. One-by-one, a column is added to the orCA and the next set of admissible columns is generated. In case $(0, 0, 0, 1)^T$ is the first column it is not possible to add another column that adheres to the coverage condition of an orCA, therefore the algorithm backtracks and replaces the first column with the next column in the list *columns*, that is $(0, 0, 1, 1)^T$. The algorithm continues with depth-first search, generating a new set of admissible columns at every recursion level and branching accordingly until finally, at recursion depth four, a fourth column is successfully added to the array, resulting in the following orCA:

(1, 2)	(2, 2)	(1, 1)	(2, 1)
1	2	3	4
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Because an orCA of the desired size is found, the algorithm immediately terminates and no further columns in the *columns* lists of the different recursion depths are examined. For example the column $(0, 0, 0, 0)^T$ in the initially computed column list is never added to an array.

4.1 Finding admissible columns for extension

In order for a column to be admissible for extension to the "current array" A, it must be ensured that the resulting array is still an orCA, *i.e.*, all relevant t-way interactions are covered if the column is added to the array. Since in general there are many possibilities for the array to be extended, we additionally apply symmetry breaking to reduce the search space. Second, the column has to adhere to some (partial) symmetry breaking constraints which are encoded in the SAT formula, in order to reduce the effort for the later applied feasibility check, which only accepts the column that results in the lexicographic smallest orCA up to *equivalence*. For CAs permutations of rows, columns or permuting the symbols of a column leaves their defining property invariant. Therefore, CAs obtained via these actions are called *equivalent* in the literature, see also [8] or [14] for more details. Similarly, the defining properties of orCAs remain invariant under permutations of rows and permutations of symbols within columns. Due to the different coverage requirements it is in general not possible to freely permute columns. Nevertheless, it is still possible to permute the s-blocks.

The symmetries induced by these actions can be removed with appropriate SAT encodings, which we describe more closely below. However, the combination of the actions mentioned above lead to additional symmetries which are not removed by the constraints encoded in the SAT formula. Hence, finding all columns admissible for extension (line 4 of Algorithm 4.1) is performed in two steps. First, a call to a SAT solver returns all columns that achieve coverage of all t-way interactions of interest and obey to the partial symmetry breaking constraints. Second, a lex-leader feasibility check removes remaining equivalent solutions, as it only accepts the column that results in the lexicographic smallest orCA in its equivalence class. The partial symmetry breaking via the SAT formula reduces the effort for the later, computationally more costly, feasibility check. The process of finding the set of admissible columns is exemplified in Example 4.2.

In the following we elaborate more detailed on the SAT formula used to find possible assignments for the k-th column of an $\mathsf{orCA}(N; t, m, s, 2)$, *i.e.*, the column with label $\varphi_{m,s}^{-1}(k)$. We consider the extension of an array $A = (a_{i,j})$ for $i \in \{1, \ldots, N\}, j \in \{1, \ldots, k-1\}$ with a k-th column, where column j of the array A is labeled with $\varphi_{m,s}^{-1}(j)$ for $j = 1, \ldots, k-1$. For this it is helpful to define the set of all $(t-1)$-way interactions, where the set of involved columns is right-justified when adding column $\varphi_{m,s}^{-1}(k)$ and their column index obtained via $\varphi_{m,s}$ is smaller than k. More specifically, let

$$\begin{aligned}\mathbb{T}_{k-1,t-1}^{or} = \{\{(p_1, v_1), \ldots, (p_{t-1}, v_{t-1})\} :& \\ & 1 \leq \varphi_{m,s}(p_i) < k \text{ for } i = 1, \ldots, t-1, \\ & \text{and } \{\varphi_{m,s}^{-1}(k), p_1, \ldots, p_{t-1}\} \text{ is right-justified}\}.\end{aligned}$$

Further, let $A\upharpoonright_\tau$ denote the set of indices of rows of A that cover the interaction τ, *i.e.*, for $\tau = \{(p_1, v_1), \ldots, (p_t, v_t)\}$ let

$$A\upharpoonright_\tau = \big\{r : a_{r,p_i} = v_i, \text{ for all } i \in \{1, \ldots, t\}\big\}.$$

The propositional formula used to derive admissible columns for extending the considered array with a k-th column consists of the following clauses:

$$\bigvee_{r \in A\upharpoonright_{\tau_{t-1}}} x_r, \qquad \text{for every } \tau_{t-1} \in \mathbb{T}^{or}_{k-1,t-1}, \tag{2}$$

$$\bigvee_{r \in A\upharpoonright_{\tau_{t-1}}} \neg x_r, \qquad \text{for every } \tau_{t-1} \in \mathbb{T}^{or}_{k-1,t-1}, \tag{3}$$

$$\neg x_r \vee x_{r+1}, \qquad \text{for each } r \text{ where } (a_{r,j})_{j=1}^{k-1} = (a_{r+1,j})_{j=1}^{k-1}, \tag{4}$$

$$x_r \vee \bigvee_{\substack{1 \le r' \le r-1,\\ a_{r',k-1}=0}} x_{r'}, \qquad \text{if } k \le m,\ \forall r \in \{1, \ldots, N\} \text{ with } a_{r,k-1} = 1, \tag{5}$$

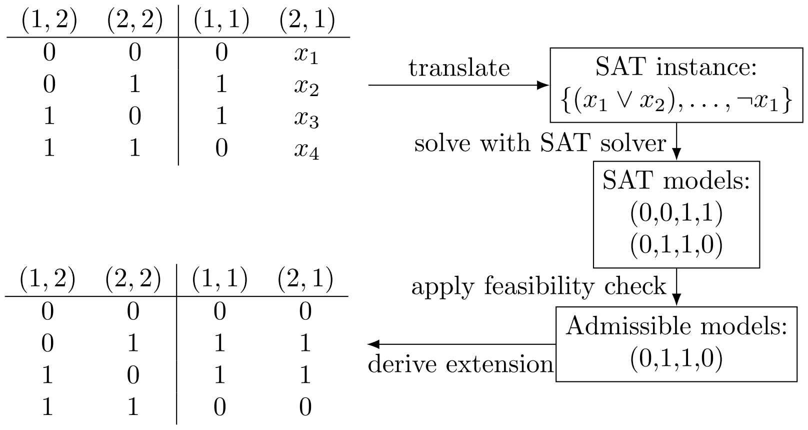
$$x_1, \qquad \text{the clause consisting of the singleton } x_1. \tag{6}$$

As mentioned above, the propositional formula incorporates clauses ensuring coverage of all required t-way interactions (clauses (2) and (3)), together with certain clauses for symmetry breaking: Additionally, we use clauses ensuring that all rows are ordered lexicographically increasing (clauses (4)). Further, the clauses in (5) ensure that the last column of every s-block (the column with index $\varphi_{m,s}(i, s)$), *i.e.*, the first column in the s-block that is considered by our algorithm, is lexicographically larger than the last column of the previous s-block (the column with index $\varphi_{m,s}(i-1, s)$). Since $\varphi_{m,s}(i-1, s) = \varphi_{m,s}(i, s) - 1$ and $\varphi_{m,s}(i, s) \le m$ for all $i \in \{2, \ldots, m\}$, the clauses in (5) enforce that the k-th column is lexicographically larger than the $(k-1)$-th column if $k \le m$. To explain these clauses, recall that $a_{r,k-1}$ denotes the value in row r of the $(k-1)$-th column considered by the algorithm, *i.e.*, column $\varphi^{-1}_{m,s}(k-1)$. For every row index r with $a_{r,k-1} = 1$ a clause is added, ensuring that either $a_{r,k} \ge a_{r,k-1} = 1$ or there exists a row with lower index $r' < r$ and $a_{r',k} < a_{r,k}$, *i.e.*, $a_{r,k} = 1$ and $a_{r,k-1} = 0$. Finally, to break symmetries resulting from symbol permutations within a column, we set the first row of every column to the symbol '0' (clause (6)). There are two main differences to the formula used in [14] for binary CAs: First, in clauses (2) and (3) coverage is only enforced for t-way interactions on right-justified sets of columns. Second, the clauses for column lexicographic order are only added for the first m columns generated by the algorithm, which become the last columns of every s-block of the final orCA. We omit more details on the derivation of the SAT-formulas and refer the interested reader to [14] for a closer explanation.

As mentioned before the symmetry breaking clauses included in the propositional formula are not sufficient to break all symmetries, therefore an additional feasibility check is conducted on every potential column for extension, *i.e.*, on every column that is a solution to the propositional formula. For that we adapt the feasibility check that is also used in [14], such that only permutations of s-blocks are considered instead of arbitrary column permutations. The feasibility check is executed for every candidate column to be added to the array A and works as follows: A linear ordering on all arrays with N rows is defined. In our case, we linearize the arrays by column-major order and compare the linearized arrays lexicographically. With respect to this linear ordering there is a smallest array in every equivalence class. The feasibility check refuses a candidate column c in case the extension of A with column c is not the smallest array in its equivalence class. For this, all orCAs that can be obtained via s-block permutations or symbol permutations within a column are generated and for each of those generated arrays the rows are sorted lexicographically. If any of those permutations leads to a lexicographically smaller array than A extended with the column c, then c is discarded as a candidate row.

Example 4.2. Assume we want to extend the 4×3 array in the left upper corner of Figure 4.1 with a column, such that the resulting array constitutes an $\mathsf{orCA}(4; 2, 2, 2, 2)$, where the columns are indexed according to $\varphi_{2,2}$. This corresponds to the step in Example 4.1 where a fourth column (*i.e.*, a column labeled by $\varphi_{2,2}^{-1}(4) = (2, 1)$) is added to the array. A SAT formula encoding coverage requirements and the partial symmetry breaking constraints is generated, for example the clause $(x_1 \vee x_2)$ ensures that one of the first two rows contains the symbol '1' and therefore coverage of the 2-way interaction $\{((1, 2), 0), ((2, 1), 1)\}$ is enforced for every solution of the SAT formula. Using a SAT solver, all solutions for the generated formula are found, in this case there are only two solutions: $(0, 0, 1, 1)$ and $(0, 1, 1, 0)$. The feasibility check executed in a second step determines that only $(0, 1, 1, 0)$ is an allowed solution, and $(0, 0, 1, 1)$ is deleted from the solution list, as it results in an orCA that is not lexicographically smallest in its equivalence class. Finally, the algorithm branches for every remaining solution and derives an orCA column from it. In the given example only the solution $(0, 1, 1, 0)$ is available and the algorithm replaces the variables in the column with label (2,1) of the orCA with appropriate values from the solution.

Using the exact algorithm described above, we can determine the minimal number of rows for which an orCA exists for given parameters as follows: If for some N, t, m, s and v an $\mathsf{orCA}(N; t, m, s, v)$ exists and the exact algorithm determines that no $\mathsf{orCA}(N - 1; t, m, s, v)$ exists, then



$(1,2)$	$(2,2)$	$(1,1)$	$(2,1)$
0	0	0	x_1
0	1	1	x_2
1	0	1	x_3
1	1	0	x_4

translate
SAT instance: $\{(x_1 \vee x_2), \ldots, \neg x_1\}$
solve with SAT solver
SAT models: (0,0,1,1) (0,1,1,0)
apply feasibility check
Admissible models: (0,1,1,0)
derive extension

$(1,2)$	$(2,2)$	$(1,1)$	$(2,1)$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Figure 4.1: Finding the set of admissible columns for the fourth column of an $\mathsf{orCA}(4; 2, 2, 2, 2)$.

$\mathsf{orCAN}(t, m, s, v) = N$. By executing the exact algorithm for different values of N, it is hence possible to determine the exact value of orCANs. For covering arrays, this strategy was applied in several works to prove optimality of some CAN bounds [11, 12].

5 Computational results

We implemented our algorithm in C++ and executed it on a machine with an AMD EPYC 7502P processor with 32 cores at 2.5 GHz base clock and 3.35 GHz boost clock and 128 GB of RAM. For computing the admissible columns in the procedure in line 4 of Algorithm 4.1 we use clasp 3.3.6 [9] for solving the occurring SAT problems.

Below we present the results of our computations. First we present tables that give the sizes of the newly found optimal orCAs. Afterwards we present and discuss bounds for covering codes in NRT spaces that can be derived using the newly found orCAs. For each individual computation we specified a time limit of two weeks.

5.1 New optimal ordered covering arrays

In Tables 5.1–5.3 we display the computational results determined with our implementation of Algorithm 4.1. Table 5.1 shows results for $\mathsf{orCAN}(t, m, s, 2)$ for strength $t = 3$, Table 5.2 shows results for strength $t = 4$ and Table 5.3 for $t = 5$. We performed experiments for s from 2 to $t - 1$ and for m starting from 3. For the considered range of s, recall Remark 2.7, which says that it is sufficient to focus on the case $s < t$, because $\mathsf{orCAN}(t, m, s, v) = \mathsf{orCAN}(t, m, t - 1, v)$ for $s \geq t$. We can consider $m \geq 3$, since in [4] for $s \geq 3$, v a prime power and $m \leq v + 1$ orCAN was determined as $\mathsf{orCAN}(s, m, s, v) = v^s$, and with $v^t \leq \mathsf{orCAN}(t, m, s, v) \leq \mathsf{orCAN}(t, m, t, v)$ we get $\mathsf{orCAN}(t, m, s, 2) = 2^t$ for all s when $t \geq 3$ and $m \leq 3$. We also added table entries for $s = 1$, where this special case corresponds to CAs. The corresponding CAN values are known and can be found, for example in [8] or [14]. A table entry for t, m, s contains the value $\mathsf{orCAN}(t, m, s, 2)$ or, in case we were only able to derive a lower bound on orCAN, an entry '$\geq N$' for a value $N \leq \mathsf{orCAN}(t, m, s, 2)$. Each orCAN entry N in the tables is derived from two results of our algorithm: a positive result for orCA-EXT$(\emptyset, N, t, m, s)$, *i.e.*, an $\mathsf{orCA}(N; t, m, s, 2)$ was found, and a negative result for orCA-EXT$(\emptyset, N - 1, t, m, s)$, *i.e.*, the algorithm determined that no $\mathsf{orCA}(N - 1, t, m, s, 2)$ exists. We display orCANs already known in the literature (from [4]) as normal text—the entries in the first column of the respective table—while new results found by our algorithm are displayed as **bold text**. For some instances, for example $m = 15$ in Table 5.1, our algorithm does not terminate within the time limit of two weeks and we are only able to give a lower bound on the corresponding orCAN. Such a lower bound $\geq N$ is derived from a negative algorithm result for orCA-EXT$(\emptyset, N - 1, t, m, s)$. The generated orCAs are available for download at [19].

Table 5.1 shows that for m from 3 to 13 the value of $\mathsf{orCAN}(3, m, 2, 2)$ is equal to $\mathsf{CAN}(3, m + 1, 2)$. For any m, an $\mathsf{orCA}(N; 3, m, 2, 2)$ with $N = \mathsf{CAN}(3, m + 1, 2)$ rows can be constructed with the construction given in the proof of [15, Theorem 4.3.1]. However, for $m = 14$ our algorithm finds a smaller orCA: $\mathsf{orCAN}(3, 14, 2, 2) = 16$, while $\mathsf{CAN}(3, 15, 2) \geq 17$ and hence [15, Theorem 4.3.1] yields an $\mathsf{orCA}(17; 3, 14, 2, 2)$. This shows that the upper bound following from the construction in [15, Theorem 4.3.1] is not tight. For $t = 3, s = 2, m = 15$ and $N = 17$ our algorithm does not terminate within the time limit. However, we know that no $\mathsf{orCA}(16; 3, 15, 2, 2)$ exists.

In Table 5.2 our orCAN results for $t = 4$ are shown. Here we can see that orCAN is not directly related to CAN. While $\mathsf{orCAN}(4, m, s, 2) =$

Table 5.1: Minimal number of rows of binary orCAs of strength three for given values of m and s: $\mathsf{orCAN}(t = 3, m, s, v = 2)$.

s \ m	3	4	5	6	7	8	9	10	11	12	13	14	15
1	8	8	10	12	12	12	12	12	12	15	16	16	17
2	8	**10**	**12**	**12**	**12**	**12**	**12**	**12**	**15**	**16**	**16**	**16**	**≥17**

$\mathsf{CAN}(4, m + 2, 2)$ for $m = 3, 5, 6$, this relationship neither holds for $m = 4$ nor $m = 7$. Interestingly, the rows for different s all show the same values, that means for the computed instances whenever an $\mathsf{orCA}(N; 4, m, 2, 2)$ exists, there is also an $\mathsf{orCA}(N; 4, m, 3, 2)$. Currently, we neither have a theoretical explanation for this, nor found an explanation in the literature. We expect this behavior to change for larger instances, but unfortunately we are not able to compute such large instances due to scalability issues of our algorithm.

Table 5.2: Minimal number of rows of binary orCAs of strength four for given values of m and s: $\mathsf{orCAN}(t = 4, m, s, v = 2)$.

s \ m	3	4	5	6	7	8	9
1	-	16	16	21	24	24	24
2	16	**22**	**24**	**24**	**27**	**≥27**	**≥27**
3	16	**22**	**24**	**24**	**27**	**≥27**	**≥27**

The same observation can be made for $t = 5$, see Table 5.3, however, in this case we are not able to compute any new values within the time limit. While our algorithm finds the (already known) minimal orCA for $m = 3$, for $m = 4$ we are not able to find an orCA and only manage to compute a lower bound of 43, *i.e.*, $\mathsf{orCAN}(5, 4, s, 2) \geq 43$.

Our computational search results show that we can find new optimal orCAs, with the presented exact column extension algorithm. These results also show that some known bounds on the size of optimal orCAs are not tight. Further, our results for $t = 4$ show that there is at least no obvious relation between orCAN and CAN that holds in general.

Table 5.3: Minimal number of rows of binary orCAs of strength five for given values of m and s: $\mathsf{orCAN}(t = 5, m, s, v = 2)$.

s \ m	3	4	5
1	-	-	32
2	32	**≥43**	**≥43**
3	32	**≥43**	**≥43**
4	32	**≥43**	**≥43**

5.2 Derived bounds on covering codes in NRT spaces

The computed optimal orCAs allow to construct covering codes in NRT spaces. We briefly revisit the most important notions pertaining covering codes in NRT spaces from [4]. Given a partially ordered set $([m \cdot s], \preceq)$ consisting of m disjoint chains with s elements each, *i.e.*, $[m \cdot s] := \{1, \dots, ms\}$, and $(i-1)s+1 \preceq \dots \preceq is$ for $i = 1, \dots, m$, the NRT metric $d_{\mathcal{R}}$ is defined on $\mathbb{Z}_q^{ms}$ as

$$d_{\mathcal{R}}(x, y) = |\langle supp(x - y)\rangle| = |\langle\{i : x_i \neq y_i\}\rangle|,$$

where $\langle A\rangle$ denotes the ideal of smallest cardinality that contains A for $A \subseteq [m \cdot s]$. An NRT space can be defined as $\mathbb{Z}_q^{ms}$ equipped with the NRT metric. A covering code with radius R in an NRT space $\mathbb{Z}_q^{ms}$ is a subset $C \subseteq \mathbb{Z}_q^{ms}$ with the property that for every $x \in \mathbb{Z}_q^{ms}$ there exists an element $c \in C$ with $d_{\mathcal{R}}(x, c) \leq R$. The smallest cardinality of a covering code with radius R of $\mathbb{Z}_q^{ms}$ is denoted by $K_q^{\mathcal{R}}(m, s, R)$. For more details on covering codes in NRT spaces, we refer the reader to [4] or [1]. Using a combinatorial construction presented in [4], which has been used in previous works to improve bounds for some covering codes in NRT spaces, it is possible to construct new covering codes in NRT spaces from orCAs and other covering codes in NRT spaces over a smaller alphabet. The authors of [4] use this construction to establish the following bound on covering codes in NRT spaces:

Theorem 5.1 (A.G. Castoldi *et al.*, [4, Theorem 16])**.** *Let v, q, m, s, R be positive integers such that $0 < R < ms$. Then*

$$K_{vq}^{\mathcal{R}}(m, s, R) \leq \mathsf{orCAN}(ms - R, m, s, v) K_q^{\mathcal{R}}(m, s, R). \tag{7}$$

Clearly, provided the respective structures exist, this result yields an upper bound on $K_{vq}^{\mathcal{R}}(m, s, R)$. However, for some of the cases presented below we only know the exact value of $\mathsf{orCAN}(ms-R,m,s,v)$, while the appearing $K_q^{\mathcal{R}}(m,s,R)$ is not known (to us). In this case we still denote the upper bound on $K_{vq}^{\mathcal{R}}(m,s,R)$ as a function of $K_q^{\mathcal{R}}(m,s,R)$. In Table 5.4 we present the inequalities that can be derived using our orCAN results presented in the previous subsection, in combination with [4, Theorem 16]. In the column '*ub*' an upper bound for $K_{2q}^{\mathcal{R}}(m,s,R)$ is given, where m, s and R are given in the respective table columns and $q \geq 2$ is an arbitrary integer. The listed upper bounds are derived from an orCAN, multiplied with a covering code size $K_q^{\mathcal{R}}(m,s,R)$. In the right-most column of the table, headed by 'Used orCAN result', the orCANs required for computing the bounds are listed. Additionally, we note that Table 5.4 could be extended with further rows, as follows. As stated in Remark 2.7, for $s \geq t$ we have $\mathsf{orCAN}(t,m,s,v) = \mathsf{orCAN}(t,m,t-1,v)$. Thus, our orCAN results can be used to derive additional upper bounds on $K_{2q}^{\mathcal{R}}(m,s,ms-t)$:

$$K_{2q}^{\mathcal{R}}(m,s,ms-t) \leq \mathsf{orCAN}(t,m,t-1,2)K_q^{\mathcal{R}}(m,s,ms-t),$$

yielding additional rows in Table 5.4 for arbitrary $t \geq 2$, all $s \geq t$ and m where $\mathsf{orCAN}(t,m,t-1,2)$ is known. For some values $K_q^{\mathcal{R}}(m,s,R)$ in (7) there exist explicit bounds in the literature. For $q=2$ explicit bounds can be found in [1], which we can use to derive explicit bounds for $K_4^{\mathcal{R}}(m,s,R)$ applying inequality (7). We summarize the thus obtained bounds in Table 5.5, and further discuss them below. Similarly to Table 5.4, an upper bound for $K_4^{\mathcal{R}}(m,s,R)$ is given in the column headed by '*ub*', for m,s,R as specified in the respective table columns. The upper bound is calculated via [4, Theorem 16] from the orCANs given in the column 'used orCAN result', together with the exact value or upper bound of $K_2^{\mathcal{R}}(m,s,R)$ given in column 'used CC bound', which can be found in the source specified in column 'Reference'.

5.3 Upper bounds on $K_4^{\mathcal{R}}(m,s,ms-t)$: contextualization to existing results

Assessing the quality of the obtained upper bounds on covering codes in NRT spaces given in Table 5.5, is not straight forward, as to the best of our knowledge there does not exist a resource providing a comprehensive overview. For the binary case several bounds can be found in [1], however, for the non-binary case such a reference seems to be absent. This also explains why Table 5.5 is only given for $q=4$. In order to get an understanding of the quality of the derived upper bounds on covering codes in NRT spaces, we have examined the—to the best of our knowledge—entire

Table 5.4: Bounds on the size $K_{2q}^{\mathcal{R}}(m,s,R)$ of covering codes in the NRT space $(\mathbb{Z}_{2q}^{ms}, d_{\mathcal{R}})$, for arbitrary $q \in \mathbb{N}$ and m, s, R.

m	s	R	*ub*	Used orCAN result
4	2	5	$10K_q^{\mathcal{R}}(4,2,5)$	$\mathsf{orCAN}(3,4,2,2) = 10$
5	2	7	$12K_q^{\mathcal{R}}(5,2,7)$	$\mathsf{orCAN}(3,5,2,2) = 12$
6	2	9	$12K_q^{\mathcal{R}}(6,2,9)$	$\mathsf{orCAN}(3,6,2,2) = 12$
7	2	11	$12K_q^{\mathcal{R}}(7,2,11)$	$\mathsf{orCAN}(3,7,2,2) = 12$
8	2	13	$12K_q^{\mathcal{R}}(8,2,13)$	$\mathsf{orCAN}(3,8,2,2) = 12$
9	2	15	$12K_q^{\mathcal{R}}(9,2,15)$	$\mathsf{orCAN}(3,9,2,2) = 12$
10	2	17	$12K_q^{\mathcal{R}}(10,2,17)$	$\mathsf{orCAN}(3,10,2,2) = 12$
11	2	19	$15K_q^{\mathcal{R}}(11,2,19)$	$\mathsf{orCAN}(3,11,2,2) = 15$
12	2	21	$16K_q^{\mathcal{R}}(12,2,21)$	$\mathsf{orCAN}(3,12,2,2) = 16$
13	2	23	$16K_q^{\mathcal{R}}(13,2,23)$	$\mathsf{orCAN}(3,13,2,2) = 16$
14	2	25	$16K_q^{\mathcal{R}}(14,2,25)$	$\mathsf{orCAN}(3,14,2,2) = 16$
4	2	4	$22K_q^{\mathcal{R}}(4,2,4)$	$\mathsf{orCAN}(4,4,2,2) = 22$
5	2	6	$24K_q^{\mathcal{R}}(5,2,6)$	$\mathsf{orCAN}(4,5,2,2) = 24$
6	2	8	$24K_q^{\mathcal{R}}(6,2,8)$	$\mathsf{orCAN}(4,6,2,2) = 24$
7	2	10	$27K_q^{\mathcal{R}}(7,2,10)$	$\mathsf{orCAN}(4,7,2,2) = 27$
4	3	8	$22K_q^{\mathcal{R}}(4,3,8)$	$\mathsf{orCAN}(4,4,3,2) = 22$
5	3	11	$24K_q^{\mathcal{R}}(5,3,11)$	$\mathsf{orCAN}(4,5,3,2) = 24$
6	3	14	$24K_q^{\mathcal{R}}(6,3,14)$	$\mathsf{orCAN}(4,6,3,2) = 24$
7	3	17	$27K_q^{\mathcal{R}}(7,3,17)$	$\mathsf{orCAN}(4,7,3,2) = 27$

Table 5.5: Bounds on the size $K_4^{\mathcal{R}}(m,s,R)$ of covering codes in the NRT space $(\mathbb{Z}_4^{ms}, d_{\mathcal{R}})$, for m, s and R as specified in the respective columns.

m	s	R	*ub*	used orCAN result	used CC bound	Reference
4	2	5	30	$\mathsf{orCAN}(3,4,2,2) = 10$	$K_2^{\mathcal{R}}(4,2,5) \leq 3$	[1, Tab. 5]
5	2	7	24	$\mathsf{orCAN}(3,5,2,2) = 12$	$K_2^{\mathcal{R}}(5,2,7) = 2$	[1, Thm. 13]
4	3	9	30	$\mathsf{orCAN}(3,4,3,2) = 10$	$K_2^{\mathcal{R}}(4,3,9) \leq 3$	[1, Tab. 6]
5	3	12	24	$\mathsf{orCAN}(3,5,3,2) = 12$	$K_2^{\mathcal{R}}(5,3,12) = 2$	[1, Thm. 13]
4	4	13	30	$\mathsf{orCAN}(3,4,4,2) = 10$	$K_2^{\mathcal{R}}(4,4,13) \leq 3$	[1, Tab. 6]
5	4	17	24	$\mathsf{orCAN}(3,5,4,2) = 12$	$K_2^{\mathcal{R}}(5,4,17) = 2$	[1, Thm. 13]
4	2	4	176	$\mathsf{orCAN}(4,4,2,2) = 22$	$K_2^{\mathcal{R}}(4,2,4) \leq 8$	[1, Tab. 5]
4	3	8	154	$\mathsf{orCAN}(4,4,3,2) = 22$	$K_2^{\mathcal{R}}(4,3,8) \leq 7$	[1, Tab. 6]
4	4	12	154	$\mathsf{orCAN}(4,4,4,2) = 22$	$K_2^{\mathcal{R}}(4,4,12) \leq 7$	[1, Tab. 6]

related literature ([1], [3], [4], [2]) on covering codes in NRT spaces. Although these works are limited in number, there are numerous constructions and bounds presented therein, which can be combined in manifold ways. In Table 5.6 we compare our bounds with other bounds, where most of them require a combination of more than one result, for example they rely on other, smaller covering codes in NRT spaces. Only for some bounds, such as the *trivial upper bound* (8), the value can be computed directly. Below we list the inequalities found in the literature we used to compute the upper bounds on covering codes in NRT spaces given in Table 5.6:

The *trivial upper bound* as also given in [1, Proposition 6] gives a bound for $m, s, q \geq 2$ and $0 \leq R \leq ms$:

$$K_q^{\mathcal{R}}(m,s,R) \leq q^{ms-R}, \tag{8}$$

the *directed sum bound* from [1, Theorem 15]: for $R_1 \leq m_1 s$ and $R_2 \leq m_2 s$

$$K_q^{\mathcal{R}}(m_1+m_2,s,R_1+R_2) \leq K_q^{\mathcal{R}}(m_1,s,R_1)K_q^{\mathcal{R}}(m_2,s,R_2), \tag{9}$$

and the following bounds for $n \leq m$ and $R \leq ns$ found in the respective references:

$$K_q^{\mathcal{R}}(m,s,R) \leq q^{(m-n)s}K_q^{\mathcal{R}}(n,s,R), \qquad \text{see [1, Corollary 16]} \tag{10}$$

$$K_q^{\mathcal{R}}(m,s,R) \leq K_q^{\mathcal{R}}(m-n,s,R-ns), \qquad \text{see [1, Proposition 17]} \tag{11}$$

the bound found in [1, Proposition 18], for $r < s$ and $R \leq mr$:

$$K_q^{\mathcal{R}}(m,s,R) \leq q^{m(s-r)}K_q^{\mathcal{R}}(m,r,R), \tag{12}$$

from [1, Proposition 21], for $r < s$ and $R \leq mr$:

$$K_q^{\mathcal{R}}(m,s,R) \leq K_q^{\mathcal{R}}(m,s-r,R-mr), \tag{13}$$

the bound from [1, Theorem 30]: if there is an MDS code in the NRT space $\mathbb{Z}_q^{ms}$ with minimum distance $d+1$, then for every $r \geq 2$

$$K_{qr}^{\mathcal{R}}(m,s,d) \leq q^{ms-d}K_r^{\mathcal{R}}(m,s,d), \tag{14}$$

and the bound from [3, Corollary 7 (3)], for q a prime power:

$$K_{(q-1)v}^{\mathcal{R}}(q+1,t,qt) \leq (q^t-2)v^{t-2}(v^2-1). \tag{15}$$

Lastly, the *ball covering bound* is a basic lower bound for $K_q^{\mathcal{R}}(m,s,R)$, see for example [1, Proposition 7]:

$$K_q^{\mathcal{R}}(m,s,R) \geq \frac{q^{ms}}{V_q^{\mathcal{R}}(m,s,R)}, \tag{16}$$

where $V_q^{\mathcal{R}}(m, s, R)$ is the cardinality of a sphere of radius R in $(\mathbb{Z}_q^{ms}, d_{\mathcal{R}})$. Let $\Omega_j(i)$ be the number of ideals of the NRT poset $[m \cdot s]$, with cardinality i and exactly j maximal elements. Then

$$V_q^{\mathcal{R}}(m, s, R) = 1 + \sum_{i=1}^{R} \sum_{j=1}^{\min\{m,i\}} q^{i-j}(q-1)^j \Omega_j(i).$$

We use the ball covering bound to show that some existing bounds in the literature are weaker than the bounds obtained using orCANs, *i.e.*, via the inequality in (7).

The first three columns of Table 5.6 specify the covering code and the considered NRT space, where the alphabet size $q = 4$ in all cases. In the column headed by 'orCAN' we denote the bound obtained from Tables 5.1 and 5.2, together with [4, Theorem 16] and the *upper and lower bounds* of $K_2^{\mathcal{R}}(m, s, R)$ according to [1]. Hence, since [4, Theorem 16] gives an upper bound for $K_4^{\mathcal{R}}(m, s, R)$ depending on $K_2^{\mathcal{R}}(m, s, R)$, the resulting table entries give a *range* of the upper bound achievable with inequality (7). We denote this by $x - y$, where y is a guaranteed upper bound obtained via inequality (7), while x only provides an upper bound in the best case scenario that $K_2^{\mathcal{R}}(m, s, R)$ is equal to the lower bound given in [1]. Note that x does not represent a lower bound on $K_4^{\mathcal{R}}(m, s, R)$. A simple entry x in column 'orCAN' simply means that the upper bound is x. This is the case when $K_2^{\mathcal{R}}(m, s, R)$ is known. In the three columns headed by 'Weaker Bounds' we give the respective `value` and the *required input* needed to obtain this value using the inequality listed in the column with the sub-heading '`ineq`'. These are weaker bounds on $K_4^{\mathcal{R}}(m, r, R)$ than the one in column 'orCAN'. Analogously, in the columns headed by 'Tighter Bounds' we display the respective information of tighter upper bounds on $K_4^{\mathcal{R}}(m, r, R)$. The tightest bound is displayed **bold**. Finally the two columns headed by 'Other applicable Bounds' give the respective *required input* that would be needed in order to obtain an explicit value via the inequality given in the column with the sub-heading '`ineq`'. Due to the absence of the required input, or a sufficiently strong bound on it, we are not able to compute this bound.

Table 5.6 shows that the bounds on $K_4^{\mathcal{R}}(m, s, R)$ obtained through orCANs and inequality (7) are stronger than some of the bounds derived based on existing results in the literature, but in all cases it was possible to derive a stronger bound based on these results. However, in some cases there seems to be potential to improve bounds based on inequality (7), depending on the size of $K_2^{\mathcal{R}}(m, s, R)$.

Table 5.6: A comparison of existing upper bounds on $K_4^{\mathcal{R}}(m,s,R)$ for $R = ms - t$, *i.e.*, the size of covering codes in NRT spaces $(\mathbb{Z}_4^{ms}, d_{\mathcal{R}})$ with radius R, with the upper bounds derived via $\mathsf{orCA}(N; R-ms, m, s, 2)$ using [4, Theorem 16].

m	s	R	orCAN	Weaker Bounds			Tighter Bounds			Other Applicable Bounds	
			ub	value	required input	Ineq	value	required input	Ineq	required input	Ineq
4	2	5	$20-30$	≥ 32	$K_4^{\mathcal{R}}(3,2,5) \geq 2$ by (16)	(10)	**24**	$K_4^{\mathcal{R}}(4,1,1) = 24$ $K_4^{\mathcal{R}}(4,1,1) = K_4(4,1)$ [1] $K_4(4,1) = 24$ [5, Table 6.3]	(13)	$K_q^{\mathcal{R}}(m_1,s,R_1)K_q^{\mathcal{R}}(m_2,s,R_2)$	(9)
				64		(8)				$K_4^{\mathcal{R}}(3,2,3)$	(11)
				≥ 256		(12)				MDS code in $(\mathbb{Z}_4^{4\cdot 2}, d_{\mathcal{R}})$: $[4,2,3,6]_2$	(14)
5	2	7	24	64		(8)	24	$K_4^{\mathcal{R}}(4,2,5) \leq 24$	(11)	$K_q^{\mathcal{R}}(m_1,s,R_1)K_q^{\mathcal{R}}(m_2,s,R_2)$	(9)
				≥ 32	$K_4^{\mathcal{R}}(4,2,7) \geq 2$ by (16)	(10)	**16**	$K_4^{\mathcal{R}}(5,1,2) \leq 16$ $K_4^{\mathcal{R}}(5,1,2) = K_4(5,2)$ [1] $K_4(5,2) \leq 16$ [5, Table 6.3]	(13)	MDS code in $(\mathbb{Z}_4^{5\cdot 2}, d_{\mathcal{R}})$: $[5,2,3,8]_2$	(14)
				≥ 1024		(12)					
4	3	9	$20-30$	64		(8)	**24**	$K_4^{\mathcal{R}}(4,2,5) \leq 24$	(13)	$K_q^{\mathcal{R}}(m_1,s,R_1)K_q^{\mathcal{R}}(m_2,s,R_2)$	(9)
				≥ 64		(10)				$K_4^{\mathcal{R}}(3,3,6)$	(11)
				≥ 256		(12)				MDS code in $(\mathbb{Z}_4^{4\cdot 3}, d_{\mathcal{R}})$: $[4,3,3,10]_2$	(14)
				75		(15)					
5	3	12	24	64		(8)	24	$K_4^{\mathcal{R}}(4,3,9) \leq 24$	(11)	$K_q^{\mathcal{R}}(m_1,s,R_1)K_q^{\mathcal{R}}(m_2,s,R_2)$	(9)
				≥ 64		(10)	**16**	$K_4^{\mathcal{R}}(5,2,7) \leq 16$	(13)	MDS code in $(\mathbb{Z}_4^{5\cdot 3}, d_{\mathcal{R}})$: $[5,3,3,13]_2$	(14)
				≥ 1024		(12)					
4	4	13	$20-30$	64		(8)	**24**	$K_4^{\mathcal{R}}(4,3,9) \leq 24$	(13)	$K_q^{\mathcal{R}}(m_1,s,R_1)K_q^{\mathcal{R}}(m_2,s,R_2)$	(9)
				≥ 256		(10)				$K_4^{\mathcal{R}}(3,4,9)$	(11)
				≥ 256		(12)				MDS code in $(\mathbb{Z}_4^{4\cdot 4}, d_{\mathcal{R}})$: $[4,4,3,14]_2$	(14)
5	4	17	24	64		(8)	24	$K_4^{\mathcal{R}}(4,4,13) \leq 24$	(11)	$K_q^{\mathcal{R}}(m_1,s,R_1)K_q^{\mathcal{R}}(m_2,s,R_2)$	(9)
				≥ 256		(10)	**16**	$K_4^{\mathcal{R}}(5,3,12) \leq 16$	(13)	MDS code in $(\mathbb{Z}_4^{5\cdot 4}, d_{\mathcal{R}})$: $[5,4,3,18]_2$	(14)
				≥ 1024		(12)					
4	2	4	$66-176$	256		(8)	**144**	$K_4^{\mathcal{R}}(2,2,2) \leq 12$ by (7), [1, Table 3] and $\mathsf{orCAN}(2,2,2,2) = 4$ [4]	(9)	$16K_4^{\mathcal{R}}(3,2,4)$	(10)
				≥ 256		(12)				$K_4^{\mathcal{R}}(3,2,2)$	(11)
				≥ 256		(13)				MDS code in $(\mathbb{Z}_4^{4\cdot 2}, d_{\mathcal{R}})$: $[4,2,4,5]_2$	(14)
4	3	8	$66-154$	256		(8)	**144**	$K_4^{\mathcal{R}}(4,2,4) \leq 144$	(13)	$64K_4^{\mathcal{R}}(3,3,8)$	(10)
				≥ 256		(12)				$K_4^{\mathcal{R}}(3,3,5)$	(11)
										MDS code in $(\mathbb{Z}_4^{4\cdot 3}, d_{\mathcal{R}})$: $[4,3,4,9]_2$	(14)
4	4	12	$44-154$	256		(8)	**144**	$K_4^{\mathcal{R}}(4,3,8) \leq 144$	(13)	$K_4^{\mathcal{R}}(3,4,8)$	(11)
				≥ 256		(10)				MDS code in $(\mathbb{Z}_4^{4\cdot 4}, d_{\mathcal{R}})$: $[4,4,4,13]_2$	(14)
				≥ 256		(12)					
				3048		(15)					

6 Conclusion and future work

Using our exact algorithm we are able to derive 19 new orCANs, *i.e.*, the number of rows of optimal ordered CAs, for small values of t, m, s and a binary alphabet $v = 2$. Additionally, we provide lower bounds for some orCANs. Our results show that some bounds on the size of optimal orCAs are not tight, and that there exists at least no obvious relation between orCAN and CAN, *i.e.*, the size of optimal CAs. These two observations underpin that orCAs are combinatorial objects that deserve to be examined independently from closely related combinatorial designs, such as covering arrays, orthogonal arrays and ordered orthogonal arrays. Our investigation of the construction of covering codes in NRT spaces using ordered CAs shows that the newly computed optimal orCAs cannot be used to improve any upper bound on the size of covering codes in NRT spaces. However, such bounds derived from orCAs are tighter than several existing ones, or match them. In some cases there remains even potential for an improvement.

The presented algorithm performs well for small instances, however, for larger instances it does not terminate within reasonable time. In order to improve the execution time of our algorithm, for satisfiable instances, *i.e.*, parameters N, t, m and s, where an $\mathsf{orCA}(N; t, m, s, 2)$ exists, the selection of admissible columns could be improved with a guiding heuristic. For example, prioritizing columns leading to a more *balanced* array, where balanced means that all i-way interactions occur evenly often, for some $i < t$. For unsatisfiable instances, on the other hand, it would be needed to speed up solution enumeration. A possible approach to achieve this would be by search space reduction, for example with better symmetry breaking in the SAT formula, in order to reduce the number of solutions that have to be considered. In the future we intend to generalize our algorithm to higher alphabet sizes $v > 2$. Additionally, we intend to explore the applicability of *balance*, as introduced in [14] for CAs, to orCAs. While our exact algorithm allowed us to produce orCAs for some instances and lower bounds for the orCAN of some other instances, a heuristic algorithm could be of interest to produce orCAN upper bounds also for larger instances.

Acknowledgments

SBA Research (SBA-K1) is a COMET Centre within the framework of COMET—Competence Centers for Excellent Technologies Programme and funded by BMK, BMDW, and the federal state of Vienna. The COMET Programme is managed by FFG.

References

[1] A.G. Castoldi and E.L. Monte Carmelo, The covering problem in Rosenbloom-Tsfasman spaces, *Electron. J. Combin.*, **22** (2015), paper 3.30, `https://doi.org/10.37236/4974`.

[2] A.G. Castoldi, E.L. Monte Carmelo and R. da Silva, Partial sums of binomials, intersecting numbers, and the excess bound in Rosenbloom-Tsfasman space, *Comput. Appl. Math.*, **38** (2019), article 55, `https://doi.org/10.1007/s40314-019-0828-2`.

[3] A.G. Castoldi, E.L. Monte Carmelo, L. Moura, D. Panario and B. Stevens, Bounds on covering codes in RT spaces using ordered covering arrays, *Lecture Notes in Comput. Sci.*, **11545** (2019), 100–111.

[4] A.G. Castoldi, E.L. Monte Carmelo, L. Moura, D. Panario and B. Stevens, Ordered covering arrays and upper bounds on covering codes, *J. Combin. Des.*, **31** (2023), 304–329.

[5] G. Cohen, I. Honkala, S. Litsyn and A. Lobstein, *Covering codes*, Elsevier, 1997.

[6] C.J. Colbourn, Combinatorial aspects of covering arrays, *Matematiche (Catania)*, **59**(1,2) (2004), 125–172.

[7] C.J. Colbourn and J.H. Dinitz, *Handbook of combinatorial designs*, Taylor & Francis Group, CRC Press, 2007.

[8] C.J. Colbourn, G. Kéri, P.P. Rivas Soriano and J.-C. Schlage-Puchta, Covering and radius-covering arrays: Constructions and classification, *Discrete Appl. Math.*, **158** (2010), 1158–1180.

[9] M. Gebser, B. Kaufmann and T. Schaub, Conflict-driven answer set solving: From theory to practice, *Artificial Intelligence*, **187–188** (2012), 52–89.

[10] A.S. Hedayat, N.J.A. Sloane and J. Stufken, *Orthogonal arrays: theory and applications*, Springer, 2012.

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

[12] I. Izquierdo-Marquez and J. Torres-Jimenez, New optimal covering arrays using an orderly algorithm, *Discrete Math. Algorithms Appl.*, **10** (2018), 1850011.

[13] L. Kampel and D.E. Simos, A survey on the state of the art of complexity problems for covering arrays, *Theoret. Comput. Sci.*, **800** (2019), 107–124.

[14] L. Kampel, I. Hiess, I.S. Kotsireas and D.E. Simos, Balanced covering arrays: A classification of covering arrays and packing arrays via exact methods, *J. Combin. Des.*, **31** (2023), 205–261.

[15] T. Krikorian, *Combinatorial constructions of ordered orthogonal arrays and ordered covering arrays*, M.S. thesis, Department of Mathematics, Ryerson University, 2011.

[16] D.R. Kuhn, R.N. Kacker and Y. Lei, *Introduction to Combinatorial Testing*, Taylor & Francis Group, 2013.

[17] K.M. Lawrence, A combinatorial characterization of (t, m, s)-nets in base b, *J. Combin. Des.*, **4** (1996), 275–293.

[18] W.J. Martin and D.R. Stinson, A Generalized Rao Bound for Ordered Orthogonal Arrays and (t, m, s)-Nets, *Canad. Math. Bull.*, **42** (1999), 359–370.

[19] MATRIS, Webpage: Optimal Ordered Covering Arrays, https://srd.sba-research.org/data/orcas/ Accessed: 2024-03-15.

[20] G.L. Mullen and W.Ch. Schmid, An equivalence between (t, m, s)-nets and strongly orthogonal hypercubes, *J. Combin. Theory Ser. A*, **76** (1996), 164–174.

[21] T. Nanba, T. Tsuchiya and T. Kikuno, Using satisfiability solving for pairwise testing in the presence of constraints, *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, **E95-A**(9) (2012), 1501–1505.

[22] D. Panario, M. Saaltink, B. Stevens and D. Wevrick, A general construction of ordered orthogonal arrays using LFSRs, *IEEE Trans. Inform. Theory*, **65** (2019), 4316–4326.

[23] S. Raaphorst, L. Moura and B. Stevens, Variable strength covering arrays, *J. Combin. Des.*, **26** (2018), 417–438.

Irene Hiess
SBA Research, MATRIS, Vienna, Austria
ihiess@sba-research.org

Ludwig Kampel
SBA Research, MATRIS, Vienna, Austria
lkampel@sba-research.org

Dimitris E. Simos
SBA Research, MATRIS, Vienna, Austria
dsimos@sba-research.org