Integer and Constraint Programming Revisited for Mutually Orthogonal Latin Squares

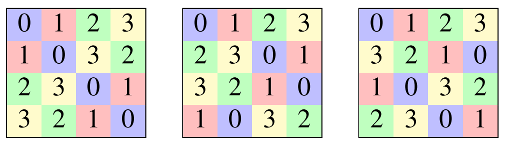
Noah Rubin[*], Curtis Bright[†][*], Kevin K. H. Cheung[*], Brett Stevens[*]
[*]Carleton University
[†]University of Windsor

***Abstract*—We use integer programming (IP) and constraint programming (CP) to search for sets of mutually orthogonal latin squares (MOLS). We improve the performance of the solvers by formulating an extended symmetry breaking method and provide an alternative CP encoding which performs much better in practice. Using state-of-the-art solvers as black boxes we are able to quickly find pairs of MOLS (or prove their nonexistence) in all orders up to and including eleven. We also analyze the effectiveness of using CP and IP solvers to search for triples of MOLS and estimate the running time of using this approach to resolve the longstanding open problem of determining the existence of a triple of MOLS of order ten.**

I. Introduction

A *latin square of order* n is an $n \times n$ array, L, of symbols $\{0, 1, \ldots, n-1\}$ in which each symbol appears exactly once in each row and column. Each of the squares in Figure 1 is an example of a latin square of order 4. The entry in row i and column j of a square L is denoted L_{ij}. Two latin squares of the same order, L and M, are said to be *orthogonal* if there is a unique solution $L_{ij} = a$, $M_{ij} = b$ for every pair of $a, b \in \{0, 1, \ldots, n-1\}$. A set of k latin squares of order n, is called a set of *mutually orthogonal latin squares* (MOLS) if all squares are pairwise orthogonal—in which case we label the system as $k\,\mathrm{MOLS}(n)$. Figure 1 shows a set of $3\,\mathrm{MOLS}(4)$. Latin squares, sets of mutually orthogonal latin squares, and a myriad of related objects have numerous applications in statistics, reliability testing, coding theory, and recreational mathematics [10], [14], [29].

There is a long history of using "automated reasoning" tools like constraint and integer programming solvers to search for and construct latin squares [18]. Since latin square problems can be easily expressed in the language of these solvers they are a popular source of benchmarks in the artificial intelligence, constraint programming, and satisfiability solving communities [21]. There has been much work developing improved solvers for such problems [13], [15], [45]. In this paper, we focus on developing improved encodings for constructing latin squares and use off-the-shelf constraint programming (CP) and integer programming (IP) solvers to find mutually orthogonal latin squares. General background for IP and CP solvers is reviewed in Section II and the IP and CP models that we use in our searches is outlined in Section III. Using IP and CP solvers on a single desktop computer we find mutually orthogonal latin squares (or disprove their existence) in all orders up to and including eleven. We also develop two main improvements to the basic models.



0	1	2	3
1	0	3	2
2	3	0	1
3	2	1	0

0	1	2	3
2	3	0	1
3	2	1	0
1	0	3	2

0	1	2	3
3	2	1	0
1	0	3	2
2	3	0	1

Fig. 1. A set of three mutually orthogonal latin squares of order four.

First, we develop an improved constraint programming encoding which we show performs significantly better than the standard constraint programming encoding—using a CP solver we were able to find or prove the nonexistence of $2\,\mathrm{MOLS}(n)$ for $n \leq 11$ with our improved encoding compared to $n \leq 8$ with the usual encoding. Our improved constraint programming formulation is described in Section III-B.

Second, we develop an improved symmetry breaking method that removes more symmetry from the search space than the "domain reduction" symmetry breaking method used in previous searches [1], [2]. We show that our symmetry breaking method reduces the amount of symmetry present in the $2\,\mathrm{MOLS}(n)$ search by an exponential factor in n when compared with domain reduction symmetry breaking. The new symmetry breaking method performs well in practice, particularly with the integer programming model—using an IP solver we were able to find or prove the nonexistence of $2\,\mathrm{MOLS}(n)$ for $n \leq 10$ with our improved symmetry breaking compared to $n \leq 8$ with domain reduction symmetry breaking. Our improved symmetry breaking is described in Section III-C.

Finally, we give detailed timings for each method in Section IV and also apply the method to search for triples of orthogonal latin squares in Section IV-B. Related work and a summary of the extensive literature on latin squares is given in Section V.

II. Background

Integer linear programming (IP) and constraint programming (CP) are two paradigms for solving combinatorial optimization and search problems. IP solvers depend heavily on numerical solutions to linear programming problems whereas CP solvers are based on search techniques such as domain reduction. The two approaches offer complementary strengths and there has been a significant amount of research on integrating the two to solve difficult combinatorial optimization search problems [23].

Both approaches use the "divide and conquer" paradigm of splitting a problem into subproblems by choosing a variable and then branching on each possible value for that variable. Each subproblem formed in this way is called a *node* of the search. A *branch* of the search is formed by assigning a fixed value to the branching variable selected in each node. IP and CP solvers typically report the number of nodes and/or the number of branches that they explore during the search.

IP solvers also solve the linear programming "relaxation" of the problem by dropping the constraint that the variables must be integers. If the relaxation has no solution over the rationals then the original problem must also be infeasible. If the relaxation has a solution then the solver examines if it violates any inequalities that can be derived from the original set of linear constraints assuming that the variables must take on integer values. Such inequalities are known as *cutting planes* and they can be effective at pruning the search space and forcing the solution of the linear program to integer values. If no cutting planes can be derived and the relaxation solution has non-integral values then a variable is chosen to branch on and new nodes are created.

Intuitively, a CP solver is effective at enumerating solutions quickly as it is not required to solve linear programs during its search. However, the relaxation solutions provided by the IP solver—despite being relatively expensive to compute—help provide a "global" perspective of the search space. In particular, Appa *et al.* present IP and CP models to search for MOLS and develop sophisticated techniques for combining solvers in ways that exploit each solver's strengths [1], [2].

In addition to pure IP/CP approaches they present ways of embedding CP into IP and vice versa. They are able to improve on an "out-of-the-box" solver using these hybrid approaches in each case when searching for $2\,\mathrm{MOLS}(n)$ with $n \leq 12$. They also extend their method to searching for triples of MOLS—using a CP solver to complete a single latin square and an IP solver to search for a pair of MOLS orthogonal to the first square. If the IP solver determines such a pair does not exist then the CP solver finds a new latin square and the process repeats. It is also possible for the CP solver to pass only a partially-completed first square to the IP solver. This makes the CP solver's search easier at the cost of making the IP solver's search more difficult and Appa *et al.* provide timings demonstrating that the runtime is typically minimized by passing a half-completed first square to the IP solver.

Moreover, Appa *et al.* analyze the symmetries present in the $k\,\mathrm{MOLS}(n)$ problem and provide effective methods of removing much of the symmetry. This is important in order to prevent solvers from exploring parts of the search space which are actually the same under some symmetry. We discuss their symmetry breaking method in detail in Section III-C and derive an improvement that we use in our work.

III. Models

In this section we describe the models that we use when searching for MOLS using integer programming (see Section III-A) and constraint programming (see Section III-B) as well as the symmetry breaking methods that we use (see Section III-C).

A. Integer Programming Model

One straightforward method of approaching the $2\,\mathrm{MOLS}(n)$ problem is to express it as a pure binary linear program and use integer programming (IP) solvers to generate solutions [11, §26.3.IV]. Let X and Y be two mutually orthogonal latin squares of order n with entries X_{ij} and Y_{ij} for $0 \leq i, j < n$. Our IP model for a set of two mutually orthogonal latin squares contains the n^4 binary variables

$$x_{ijkl} := \begin{cases} 1 & \text{if } X_{ij} = k \text{ and } Y_{ij} = l \\ 0 & \text{otherwise} \end{cases}$$

for all $i, j, k, l \in \{0, 1, \ldots, n-1\}$.

We encode the latin and orthogonality constraints as six sets of n^2 equalities grouped by which subscripts of x_{ijkl} are fixed:

$$\begin{aligned}
&\sum_{0\leq k,l<n} x_{ijkl} = 1\ \forall i,j && \text{each cell contains only 1 value}\\
&\sum_{0\leq j,l<n} x_{ijkl} = 1\ \forall i,k && \text{latin property in rows of } X\\
&\sum_{0\leq j,k<n} x_{ijkl} = 1\ \forall i,l && \text{latin property in rows of } Y\\
&\sum_{0\leq i,l<n} x_{ijkl} = 1\ \forall j,k && \text{latin property in columns of } X\\
&\sum_{0\leq i,k<n} x_{ijkl} = 1\ \forall j,l && \text{latin property in columns of } Y\\
&\sum_{0\leq i,j<n} x_{ijkl} = 1\ \forall k,l && \text{orthogonality of } X \text{ and } Y
\end{aligned}$$

This model can be extended to search for $k\,\mathrm{MOLS}(n)$ using n^{k+2} variables and $\binom{k+2}{2}n^2$ constraints, but this easily exceeds our ability to solve for all but the smallest values of k. Instead, in order to search for triples of MOLS we employ an alternative method described by Appa *et al.* [1] that relies on using a separate solver to fill in the entries of a third square Z. Assuming the entries of Z have been fixed in advance, the $3\,\mathrm{MOLS}(n)$ problem may then be represented as an extension of the above $2\,\mathrm{MOLS}(n)$ model with the following additional constraints:

$$\begin{aligned}
&\sum_{\substack{Z_{ij}=z\\ 0\leq l<n}} x_{ijkl} = 1\ \forall k,z && \text{orthogonality of } X \text{ and } Z\\
&\sum_{\substack{Z_{ij}=z\\ 0\leq k<n}} x_{ijkl} = 1\ \forall l,z && \text{orthogonality of } Y \text{ and } Z
\end{aligned}$$

B. Constraint Programming Model

A CP solver allows a more natural formulation of the $2\,\mathrm{MOLS}(n)$ problem using the $2n^2$ integer-valued variables

$$X_{ij} := \text{value of cell } (i,j) \text{ in square } X,$$
$$Y_{ij} := \text{value of cell } (i,j) \text{ in square } Y,$$

where $i, j, X_{ij}, Y_{ij} \in \{0, 1, \ldots, n-1\}$. The squares X and Y can be forced to be latin squares via "AllDifferent" constraints which specify that the rows and columns of the squares each contain different values. One natural way of encoding orthogonality (c.f. [1]) is to use a set of n^2 variables Z_{ij} defined by the linear constraints

$$Z_{ij} := X_{ij} + nY_{ij} \in \{0, 1, \ldots, n^2 - 1\}.$$

The squares X and Y are orthogonal exactly when the variables Z_{ij} contain no duplicate values and therefore the single constraint AllDifferent(Z_{ij} $\forall i, j$) ensures that X and Y are orthogonal; we call this the CP-linear encoding.

We also used an alternative formulation of ensuring orthogonality that uses no arithmetic and shorter AllDifferent constraints. In order to describe it, note that each row of a latin square may be considered as a permutation of $\{0, \ldots, n-1\}$. Denote by AB the square whose ith row consists of the composition of the permutations A_i and B_i (the ith rows of A and B). Composition is performed left-to-right following the standard convention in the field,[1] i.e., fg denotes the function $x \mapsto g(f(x))$ and so the (i, j)th entry of AB is $B_i(A_i(j))$. Additionally, A^{-1} denotes the square where each row consists of the inverse of the permutation formed by the corresponding row of A.

Our alternative orthogonality encoding is based on [29, Theorem 6.6] which implies that two latin squares X and Y are orthogonal if and only if there is a latin square Z such that $XZ = Y$. The additional variables in our alternative orthogonality encoding are

$$Z_{ij} := \text{value of cell } (i, j) \text{ in square } Z = X^{-1}Y,$$

where $Z_{ij} \in \{0, 1, \ldots, n-1\}$. In order to ensure $Y = XZ$ the (i, j)th entry of Y is set equal to the (i, X_{ij})th entry of Z. This is accomplished with the "element indexing" constraint $Z_i[X_{ij}] = Y_{ij}$ where Z_i is the vector of variables corresponding to the ith row of Z. Some CP solvers such as OR-Tools [44] have native support for such constraints via what are called "VariableElement" constraints. The constraints encoding that the squares X and Y are orthogonal are then AllDifferent(Z_{ij} $\forall j$) and AllDifferent(Z_{ij} $\forall i$). We call this the CP-index encoding and altogether it uses $6n$ AllDifferent constraints together with n^2 element indexing constraints. This encoding also may be straightforwardly extended to encode the $3\,\mathrm{MOLS}(n)$ problem by adding another n^2 variables encoding the entries of a third latin square U and using the same orthogonality constraints as above between X and U and Y and U (requiring another $2n^2$ variables Z_{ij}^{XU} and Z_{ij}^{YU}).

C. Symmetry Breaking

There are a large number of possible symmetries in the $k\,\mathrm{MOLS}(n)$ problems. In any set of $k\,\mathrm{MOLS}(n)$, the squares themselves can be permuted, the rows of all squares can be permuted simultaneously, the columns of all the squares can be permuted simultaneously, the symbol sets within each square can be permuted independently and the squares may all be replaced with their transposes. All of these operations preserve the latin properties and orthogonality of the set of squares [10]. This group of possible symmetries has order $2k!(n!)^{k+2}$ and the search space can be reduced significantly for any elimination of possible symmetries that still permits finding an isomorphic representative of any solution. Appa *et al.* fix the first row of every square to be in lexicographic order which eliminates the permutations of the columns and fixes the symbol permutations to be the same in each square. They fix the first column of the first square to be in lexicographic order which eliminates permutations of the rows. This reduces the possible symmetries to $2k!n!$. With these cells fixed, the first column of Y must be a permutation where $Y_{i0} \neq i$ for $1 \leq i < n$. The number of such permutations is approximately $(n-1)!/e$ [22].

Appa *et al.* [1], [2] further show that every possible solution is isomorphic to one satisfying $Y_{10} = 2$, $Y_{i0} \neq i$ and $Y_{i0} \leq i+1$ for $1 \leq i < n$. In other words, in addition to fixing the values of certain variables they also incorporate *domain reduction* of entries in the first column of Y. To understand the magnitude of this reduction of the search space we must count the number of these solutions. Let $P(n)$ be the set of all permutations of $\{0, 1, \ldots, n-1\}$ and $A(n)$ be

$$\{\rho \in P(n) : \rho(0) = 0,\ \rho(1) = 2,\ \rho(i) \neq i,\ \rho(i) \leq i+1\},$$

and $a(n) = |A(n)|$.

Proposition 1. *Let $F(n)$ be the nth Fibonacci number. Then $a(n) = F(n-2)$.*

Proof. Let $\rho \in A(n)$. If $\rho(n-1) = n-2$ then $\rho(i) < n-3$ for all $i < n-5$ and $\rho(n-3) \neq n-3$. Thus $\rho(n-4) = n-3$ and ρ with its last two positions deleted is in $A(n-2)$. If $\rho(n-1) \neq n-2$, then $\rho(i) < n-2$ for all $i < n-4$ so $\rho(n-3) = n-2$. In this case ρ with symbol $n-1$ deleted is in $A(n-1)$. From this we can conclude

$$a(n) \leq a(n-1) + a(n-2).$$

Given any $\rho' \in A(n-1)$ we can produce a $\rho \in A(n)$ by inserting symbol $n-1$ in the second last position. No two $\rho', \rho'' \in A(n-1)$ can produce the same $\rho \in A(n)$ this way and in every case $\rho(n-3) = n-2$. Given any $\rho' \in A(n-2)$ we can produce a $\rho \in A(n)$ by appending $n-1$ followed by $n-2$. No two $\rho', \rho'' \in A(n-2)$ can produce the same $\rho \in A(n)$ this way and in every case $\rho(n-1) = n-2$. Since symbol $n-2$ is in a different position when constructing from $A(n-1)$ or $A(n-2)$, we can conclude that

$$a(n) \geq a(n-1) + a(n-2).$$

Since $a(3) = a(4) = 1$ we conclude that $a(n) = F(n-2)$. $\square$

With this reduction in cases, Appa *et al.* have reduced the number of first columns for Y from around $(n-1)!/e$ to

$$a(n) = F(n-2) \approx \frac{\sqrt{5}}{5}\left(\frac{1+\sqrt{5}}{2}\right)^{n-2}$$

choices. This is a very significant reduction.

By exploiting the disjoint cycle structure of these permutations, we have been able to reduce this number to $b(n)$,

[1]The standard reference [29] defines function composition as performed right-to-left but in fact follows the left-to-right convention.

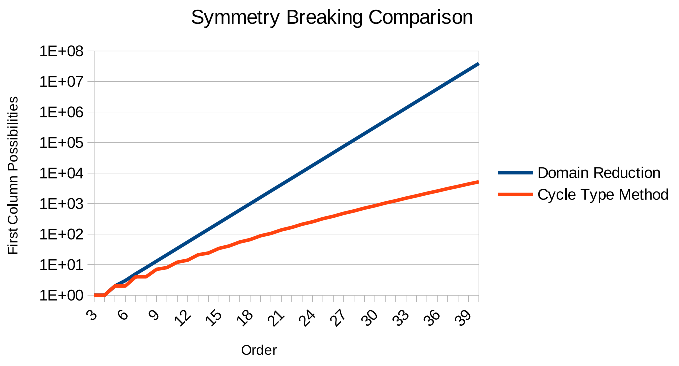


Fig. 2. A comparison between the number of possible first columns of Y using the "domain reduction" symmetry breaking of [2] with our "cycle type" method. Note that the vertical axis is logarithmic.

the number of partitions of $n-1$ into parts of size greater than 1. The values of $b(n)$ are sequence A002865 at the Online Encyclopedia of Integer Sequences and the value of $b(n)$ is approximately [41]

$$b(n) \approx \frac{\pi e^{\pi\sqrt{2(n-1)/3}}}{12\sqrt{2}(n-1)^{3/2}}.$$

This growth rate is substantially slower than the Fibonacci numbers; $b(n)$ grows exponentially slower than $a(n)$. The values of $a(n)$ and $b(n)$ for $n \le 12$ are:

n	3	4	5	6	7	8	9	10	11	12
$a(n)$	1	1	2	3	5	8	13	21	34	55
$b(n)$	1	1	2	2	4	4	7	8	12	14

Figure 2 also contains a plot comparison between $a(n)$ and $b(n)$.

We say that a pair of orthogonal latin squares X and Y of order n are in *standard form* if the first rows of both are in lexicographic order and the first column of X is in lexicographic order. The *first column permutation* of Y is the permutation of the symbols $\{1, 2, \dots, n-1\}$ defined by $\rho(i) = Y_{i0}$. The *ordered cycle* of Y is the representation of ρ as a product of disjoint cycles

$$(a_{0,0}, \dots, a_{0,l_0-1})(a_{1,0}, \dots, a_{1,l_1-1}) \cdots (a_{c,0}, \dots, a_{c,l_c-1})$$

where each cycle is the lexicographic least cyclic shift and the cycles are in lexicographic order [12]. The *ordered cycle type* of Y is $(l_0, l_1, \dots, l_c)$.

Theorem 1. *Let X and Y be a pair of orthogonal latin squares. They are isomorphic to a pair X', Y' in standard form where the ordered cycle type of Y' is non-decreasing.*

Proof. We transform X and Y into the desired form by permuting the rows and columns of both squares so that the first row and column of X are in lexicographic order. Then we permute the symbol set of Y such that the first row of Y is in lexicographic order. Let ρ be the first column permutation of this transformed Y and let $t = (l_0, l_1, \dots, l_c)$ be the ordered cycle type. Let $t' = (l_{j_0}, l_{j_1}, \dots, l_{j_c})$ be the list t sorted in non-decreasing order. Let

$$r = (a_{j_0,0}, a_{j_0,1}, \dots, a_{j_0,l_{j_0}-1}, \\ a_{j_1,0}, \dots, a_{j_1,l_{j_1}-1}, \dots, a_{j_c,0}, \dots, a_{j_c,l_{j_c}-1})$$

be a permutation of $\{1, 2, \dots, n-1\}$ in standard list form, not disjoint-cycle form. Extend r by setting $r(0) = 0$. Apply permutation r to the symbols of both X and Y, and call the resulting squares X' and Y' so $X'_{i0} = r(i)$ and $Y'_{i0} = r(Y_{i0})$. Permute the columns of both X' and Y' so the first rows of each are in lexicographic order. Now apply r^{-1} to the rows of both X' and Y' so that $X'_{i0} = r(r^{-1}(i)) = i$ and $Y'_{i0} = r(Y_{r^{-1}(i)0}) = r\rho r^{-1}(i)$, the conjugate of ρ by r. Conjugation preserves the cycle structure of a permutation and replaces each element, x of ρ in disjoint cycle notation by $r(x)$ [12]. Thus $r\rho r^{-1}$ is

$$(1, 2, \dots, l_{j_0} - 1)(l_{j_0}, \dots, l_{j_1} - 1) \cdots (l_{j_{c-1}}, \dots, l_{j_c} - 1).$$

And thus the ordered cycle type of Y' is non-decreasing. $\square$

This theorem applies equally well to a set of k MOLS(n).

Model	5	6	7	8	9	10
IP	0.07	Timeout	3.2	6.4	344.5	3,046.4
CP-linear	0.03	Timeout	3.4	81.9	1,517.0	12,515.4
CP-index	0.03	49,998.4	3.6	28.2	328.4	189.6

TABLE I
TIMINGS (IN SECONDS) FOR ORDERS $5 \le n \le 10$ WITHOUT SYMMETRY BREAKING. ALL MODELS TIMED OUT AT 60,000 SECONDS FOR $n = 11$.

IV. RESULTS

In this section we provide running times for our implementations. Unless otherwise stated, all times were recorded using a single thread on an Intel Core i9-9900K processor running at 3.6 GHz and with 32 GiB of memory. The models were compiled into executable code using Microsoft Visual C++ 2019. The IP models were solved using Gurobi version 9 [20], the CP models were solved using OR-Tools version 8 [44], and all default solver parameters were used with the exception of disabling multi-threading.

By default OR-Tools uses a branching strategy that chooses the first unassigned variable. We declared the variables so the values of the entries of X and Y are ordered by row and then by column (i.e., in the order X_{00}, Y_{00}, X_{01}, Y_{01}, …) and the variables ensuring orthogonality are ordered last. Our code is publicly available at github.com/noahrubin333/CP-IP. We first describe the times for searching for a pair of MOLS and then describe our experiments searching for a triple of MOLS.

A. 2 MOLS(n) *Timings*

As a "baseline" we first present timings for the models without any symmetry breaking included. The default symmetry breaking performed by Gurobi was also disabled for these times. In these instances the solver either reported infeasible (in order six), explicitly found a pair of MOLS, or timed out after 60,000 seconds. In Table I we compare the IP model along with the

n	Sym. Breaking	IP	CP-linear	CP-index
5	Domain Red.	0	0	0
5	(1, 2, 2)	0	0	0
5	(1, 4)	0	0	0
6	Domain Red.	2	1	0
6	(1, 5)	0	0	0
6	(1, 2, 3)	0	0	0
7	Domain Red.	6	1	0
7	(1, 2, 2, 2)	0	1	0
7	(1, 6)	0	3	0
7	(1, 2, 4)	0	4	0
7	(1, 3, 3)	0	1	0
8	Domain Red.	23,729	759	6
8	(1, 3, 4)	2	1,282	5
8	(1, 2, 2, 3)	4	293	7
8	(1, 2, 5)	4	164	13
8	(1, 7)	2	368	10
9	Domain Red.	Timeout	Timeout	290
9	(1, 3, 5)	808	Timeout	485
9	(1, 2, 2, 4)	612	Timeout	212
9	(1, 8)	303	Timeout	1,200
9	(1, 2, 6)	1,082	Timeout	61
9	(1, 2, 2, 2, 2)	1,832	Timeout	784
9	(1, 4, 4)	1,315	Timeout	917
9	(1, 2, 3, 3)	65	Timeout	223
10	Domain Red.	Timeout	Timeout	1,379
10	(1, 4, 5)	52,329	Timeout	3,892
10	(1, 2, 2, 5)	29,387	Timeout	4,383
10	(1, 2, 2, 2, 3)	6,128	Timeout	1,326
10	(1, 2, 3, 4)	15,534	Timeout	835
10	(1, 3, 6)	31,005	Timeout	3,244
10	(1, 2, 7)	Timeout	Timeout	1,413
10	(1, 3, 3, 3)	36,959	Timeout	3,886
10	(1, 9)	Timeout	Timeout	1,319
11	Domain Red.	Timeout	Timeout	29,925
11	(1, 3, 7)	Timeout	Timeout	32,815
11	(1, 2, 2, 3, 3)	Timeout	Timeout	41,665
11	(1, 3, 3, 4)	Timeout	Timeout	Timeout
11	(1, 2, 8)	Timeout	Timeout	11,524
11	(1, 2, 2, 6)	Timeout	Timeout	8,982
11	(1, 2, 2, 2, 4)	Timeout	Timeout	8,299
11	(1, 2, 2, 2, 2, 2)	Timeout	Timeout	10,206
11	(1, 2, 4, 4)	Timeout	Timeout	19,571
11	(1, 10)	Timeout	Timeout	13,210
11	(1, 4, 6)	Timeout	Timeout	40,077
11	(1, 2, 3, 5)	Timeout	Timeout	Timeout
11	(1, 5, 5)	Timeout	Timeout	7,359

TABLE II
MEDIAN IP AND CP TIMINGS (IN SECONDS) BY SYMMETRY BREAKING TYPE (EITHER DOMAIN REDUCTION OR A GIVEN CYCLE TYPE). ALL MODELS TIMED OUT AT 60,000 SECONDS FOR $n = 12$.

two CP models—one with linear constraints (CP-linear) and one with indexing constraints (CP-index). The timings show that the CP-index model generally performs the best.

In Table II we present detailed timings for the IP and CP models with the "domain reduction" symmetry breaking of Appa *et al.* [2] and the "cycle type" symmetry breaking method described in Section III-C. For the "cycle type" symmetry breaking method the first column of X was fixed in sorted order and the first column of Y randomly assigned—only subject to the constraint that $Y_{i0} \neq i$ for $i > 0$. In the IP model the appropriate variables were assigned to 1 to encode the values in the first columns of X and Y. In the CP model the variables in the first column were set by restricting their domains to a single element.

A Python script was used to determine the cycle type of the first column permutation of Y and then categorize that instance into one of the $b(n)$ cycle type equivalence classes described in Section III-C. For each cycle type of each order, three independent runs were made in order to determine a "typical" running time for that cycle type. Table II shows the result of the median run for each cycle type when the runs were sorted in terms of running time. For the IP model the cycle type symmetry breaking method performed much better than the domain reduction method. For the CP model the cycle type symmetry breaking method often performed better but there were certain cycle types whose runs completed slower than the runs using domain reduction symmetry breaking. We were unable to determine the cycle types which would perform well or poorly in advance. This makes the cycle type method particularly appropriate for a machine with many cores—as it easily admits parallelization by running an instance for each cycle type on a separate core.

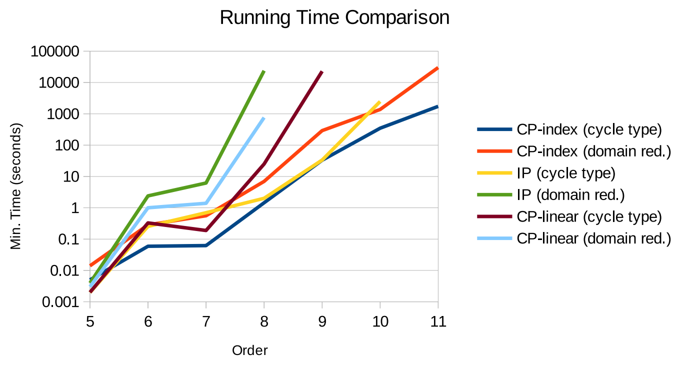


Fig. 3. A comparison of the running times for the various models and symmetry breaking methods that we considered in the orders $5 \leq n \leq 11$.

Figure 3 compares the minimum recorded running time across the orders $5 \leq n \leq 11$ for each symmetry breaking method and each model that we considered. It shows that the running time is typically exponential in n and and the CP-index model with cycle type symmetry breaking tends to have the fastest running times for constructing a latin square of order n.

B. $3\,\mathrm{MOLS}(n)$ *Timings*

Lastly, we give timings for the $3\,\mathrm{MOLS}(n)$ models presented in Section III. To implement these models we used the CP solver to enumerate latin squares of order n and these are taken as our third square Z. For each square we generate the constraint sets to encode that the third square is orthogonal to the first two squares X and Y and then solve the model using an IP or CP solver. If the model returns a feasible solution then we have found a solution to $3\,\mathrm{MOLS}(n)$, otherwise we continue by replacing the aforementioned constraints with ones generated from the next latin square produced by the CP solver. Search randomization in the CP solver was enabled so that each square was likely not too similar to the square from the previous instance.

The average time it took the solver to solve a single instance with Z fixed is given in Table III. The first column of Z was fixed in sorted order (instead of fixing the first column

Model	Sym. Breaking	6	7	8	9	10	11
IP	Domain Red.	0.0	0.7	12.8	13,418.2	Timeout	Timeout
IP	Cycle Type	0.0	0.7	9.6	Timeout	Timeout	Timeout
CP-index	Domain Red.	0.0	0.0	0.3	0.5	2.8	203.9
CP-index	Cycle Type	0.0	0.0	0.3	0.5	2.2	114.8

TABLE III
AVERAGE TIME (IN SECONDS) FOR SOLVING A $3\,\mathrm{MOLS}(n)$ INSTANCE FOR ORDER $6 \leq n \leq 11$ AND THE THIRD SQUARE FIXED. A TIMEOUT DENOTES AN INSTANCE WAS STOPPED AT 60,000 SECONDS.

Order	Cycle Type Symmetry Breaking Constraints
6	$Y_{30} = 4$
7	$Y_{40} \neq 1$
8	$Y_{50} = 6$, $Y_{40} \neq 1$
9	$Y_{60} \geq 5$, $Y_{80} \leq 5 \vee Y_{20} \neq 3$
10	$Y_{70} = 8$, $Y_{50} \in \{3, 6\}$, $Y_{60} \geq 4$, $Y_{90} \neq 7 \vee Y_{40} \neq 1$
11	$Y_{50} \neq 4$, $Y_{60} \neq 1$, $Y_{70} \in \{5, 8\}$, $Y_{80} \geq 7$, $Y_{10,0} \leq 6 \vee Y_{40} \neq 1$, $Y_{10,0} \neq 9 \vee Y_{40} \neq 5$

TABLE IV
CONSTRAINTS FOR SPECIFYING THE FIRST COLUMN OF Y IS IN CANONICAL FORM FOR ORDERS $n \leq 11$.

0	1	2	3	4	5	6	7
5	4	7	6	1	0	3	2
7	6	5	4	3	2	1	0
2	3	0	1	6	7	4	5
6	7	4	5	2	3	0	1
3	2	1	0	7	6	5	4
1	0	3	2	5	4	7	6
4	5	6	7	0	1	2	3

0	1	2	3	4	5	6	7
2	6	5	4	3	7	0	1
1	0	3	7	5	6	2	4
4	7	6	5	2	3	1	0
3	2	0	1	7	4	5	6
6	5	4	2	0	1	7	3
7	3	1	0	6	2	4	5
5	4	7	6	1	0	3	2

0	1	2	3	4	5	6	7
1	0	3	2	5	4	7	6
2	3	0	1	6	7	4	5
3	2	1	0	7	6	5	4
4	5	6	7	0	1	2	3
5	4	7	6	1	0	3	2
6	7	4	5	2	3	0	1
7	6	5	4	3	2	1	0

Fig. 4. A set of $3\,\mathrm{MOLS}(8)$ found by our search.

of X) and either domain reduction or cycle type symmetry breaking method was used on the first column of Y. Due to the superiority of the CP-index model over the CP-linear model in we only give timings for the CP-index model. These times were run on an AMD EPYC 7502P 32-Core Processor with one core solving each instance.

Cycle type symmetry breaking was encoded by selecting a canonical representative for each possible cycle type and then adding constraints to ensure the first column of Y must be canonical. For example, for $n = 6$ the columns $(0, 2, 1, 4, 5, 3)$ and $(0, 2, 3, 4, 5, 1)$ were chosen to be canonical and $(0, 2, 3, 1, 5, 4)$ was chosen to not be canonical, so we added the constraint $Y_{30} = 4$ to block the noncanonical column from occurring as the first column of Y. The IP solver was provided with linear constraints satisfied by all canonical columns but not satisfied by at least one noncanonical column. For example, for $n = 8$ we added the constraints $\sum_{k=0}^{n-1} x_{50k6} = 1$ and $\sum_{k=0}^{n-1} x_{40k1} = 0$ implying that $Y_{50} = 6$ and $Y_{40} \neq 1$. Table IV contains the constraints that we used on top of the domain reduction constraints to specify that the first column of Y is canonical.

A set of $3\,\mathrm{MOLS}(7)$ was found after 313,842 attempts and a set of $3\,\mathrm{MOLS}(8)$ was found after a single attempt. We observed that the very first latin square of order 8 generated by the CP solver (when ordering the variables by row and then by column) is highly structured[2] and part of a mutually orthogonal triple—this may explain why the timings of Appa *et al.* [1], [2] were particularly fast in order 8. See Figure 4 for the $3\,\mathrm{MOLS}(8)$ that we found with the highly structured third square. No $3\,\mathrm{MOLS}(n)$ for $n \geq 9$ were found.

V. RELATED WORK AND HISTORY

A pair of orthogonal latin squares of order 4 were known to medieval Muslim mathematicians. In 1700, the Korean mathematician Choi Seok-jeong presented $2\,\mathrm{MOLS}(9)$ and reported being unable to find a pair of order 10 [10]. Euler could construct a pair of orthogonal latin squares for all odd n and for n divisible by 4 [17]. He knew that $2\,\mathrm{MOLS}(2)$ was impossible and failed to construct a pair of order 6 using any of the methods for which he had success for other n. He verified that none of his construction methods could succeed when $n \equiv 2 \pmod 4$ and conjectured that $2\,\mathrm{MOLS}(n)$ exist if and only if $n \not\equiv 2 \pmod 4$.

The maximum number of mutually orthogonal latin squares of order n is denoted $\mathrm{N}(n)$. There do not exist $4\,\mathrm{MOLS}(4)$, so $\mathrm{N}(4) = 3$. In 1896, E. H. Moore proved that $(q-1)\,\mathrm{MOLS}(q)$ exist for every prime power q and that if $k\,\mathrm{MOLS}(n)$ and $k\,\mathrm{MOLS}(m)$ exist then $k\,\mathrm{MOLS}(mn)$ exist [36]. This proves that $\mathrm{N}(n) \geq q - 1$ where q is the smallest prime power in the prime power decomposition of n. Tarry confirmed that $2\,\mathrm{MOLS}(6)$ did not exist in 1900 [47]. In 1949, Bruck and Ryser showed that $\mathrm{N}(n) < n - 1$ if $n \equiv 1, 2 \pmod 4$ and n cannot be written as a sum of two squares [8]. Bose and Shrikhande refuted Euler's Conjecture by constructing $2\,\mathrm{MOLS}(22)$ in 1958 [5]. Two years later, Bose, Shrikhande, and Parker proved that $2\,\mathrm{MOLS}(n)$ exist for all $n \equiv 2 \pmod 4$ with $n = 2, 6$ being the only exceptions [6]. The only values of n for which it is known that $\mathrm{N}(n) = n - 1$ are prime powers [10]. In one of the most famous computational results in combinatorics, Lam, Thiel, and Swiercz showed that $\mathrm{N}(10) < 9$ [28]. This is the only known case where $\mathrm{N}(n) < n - 1$ which is not ruled out by the Bruck–Ryser–Chowla Theorem [10]. It is known that $\mathrm{N}(n) \geq n^{1/14.8}$ for n sufficiently large [4]. The Handbook of Combinatorial Designs contains tables of the best known bounds on $\mathrm{N}(n)$ in 2007 [10]. It is an open problem if $3\,\mathrm{MOLS}(10)$ exist although McKay, Meynert, and Myrvold have eliminated the possibility that such a triple could have a non-trivial automorphism group [34].

Many mathematical methods for constructing sets of mutually orthogonal latin squares exist. These range from algebraic to the recursive [10]. On the computational side, there have been exhaustive searches and non-exhaustive search using various metaheuristics. Lam, Thiel, and Swiercz's proof showing the non-existence of $9\,\mathrm{MOLS}(10)$ was an exhaustive backtracking search aided by powerful theorems from coding theory and permutation groups to eliminate finding structures isomorphic to those already found [28]. McKay, Meynert, and Myrvold use the orderly generation method to enumerate all non-isomorphic latin squares of orders up to nine [34]. They use the library `nauty` [33] to compute automorphisms and

[2]The (i, j)th entry of this square is the bitwise exclusive OR of i and j represented in binary (cf. [42]).

canonical representatives in each class. They also enumerate the different numbers of equivalence classes of these squares. McKay and Wanless are able to enumerate all latin squares of order 11 [35] at the cost of not computing the additional data reported by McKay, Meynert, and Myrvold. They proceed by generating the squares row by row using an algorithms of Sade [35], [46]. Niskanen and Östergård's clique finding software, `cliquer` has been successfully used to find mutually orthogonal latin cubes [27], [40]. Kidd [26] and Benadé, Burger, and van Vuuren [3] use custom-written backtracking searches to enumerate all triples of MOLS up to order 8. Using satisfiability (SAT) solvers with an interface to `nauty`, the nonexistence of $9\,\mathrm{MOLS}(10)$ was confirmed by Bright *et al.* [7] by producing nonexistence proof certificates.

Colbourn and Dinitz [9] implemented many of the combinatorial constructions to aid generating the tables given in the Handbook of Combinatorial Designs [10]; strictly speaking, these computational methods are not metaheuristics but neither are they exhaustive searches. Much of Colbourn and Dinitz's work has recently been implemented in the SageMath open source mathematical software system [48]. Elliott and Gibbons [16] use the *simulated annealing* metaheuristic to construct latin squares of orders up to 18 which contain no subsquares. Magos [31] has used the Tabu metaheuristic to construct latin squares. Mariot *et al.* [32] have used evolutionary algorithms and cellular automata to generate orthogonal latin squares.

An alternative approach to constructing latin squares is to formulate the problem in a declarative way and use an automated reasoning solver to search for feasible solutions. For example, Moura [37], [38] uses integer programming (IP) to search for certain combinatorial designs that are related to latin squares. Huang *et al.* [24] use constraint programming (CP) to search for orthogonal golf designs. FeiFei and Jian [30] use a first-order logic model generator to search for orthogonal latin squares while Zaikin and Kochemazov [49] use a propositional logic solver (i.e., SAT solver) to search for orthogonal diagonal latin squares and Jin *et al.* [25] use a SAT solver to search for Costas latin squares. Gomes, Regis, and Shmoys [19] employ a hybrid CP/IP approach to the problem of completing partial latin squares and Appa, Mourtos, and Magos [1], [2] investigate using IP and CP algorithms for generating pairs and triples of mutually orthogonal latin squares. They report encouraging results and propose that a hybrid IP/CP strategy of integrating the two techniques might have some success at finding $3\,\mathrm{MOLS}(10)$ if such a set exists.

VI. Conclusion

In this paper we use integer and constraint programming solvers in the search for mutually orthogonal latin squares. We demonstrate that modern state-of-the-art solvers are able to find pairs of MOLS (or demonstrate their nonexistence) in a reasonable amount of time for all orders $n \leq 11$ but struggle with higher orders. We continue the work of Appa *et al.* [1] by extending their symmetry breaking method and providing an alternative CP formulation that performs better in our implementation.

We also examine the feasibility of using CP/IP solvers in the search for triples of MOLS. One of the motivations of this work was to examine the feasibility of using CP and IP solvers in the search for a triple of MOLS of order ten. We were encouraged by the estimations presented by Mourtos [39] who fixed a pattern of zeros in one of the squares and reported that it is possible to determine the nonexistence of a triple of MOLS containing that specific pattern of zeros in about 12.5 hours. After applying symmetry breaking there are $9!$ ways of fixing the zeros in one of the squares and this results in an estimate of about 520 years of computing time in order to resolve the question of the existence of $3\,\mathrm{MOLS}(10)$. Unfortunately, our results find this estimate to be much too optimistic and by personal communication with Mourtos we were informed that the previous timings included additional restrictions of the search space that were undocumented. We were unable to rule out even a single fixing of zeros from appearing in a set of $3\,\mathrm{MOLS}(10)$. Our CP implementation with our "cycle type" symmetry breaking method required an average of 2.2 seconds to show that a fixed latin square could not be part of a set of $3\,\mathrm{MOLS}(10)$. Given that there are about $7.5 \cdot 10^{24}$ possibilities [43] for a fixed latin square of order ten (after symmetry breaking) this method would require an estimated $5 \cdot 10^{17}$ years to resolve the $3\,\mathrm{MOLS}(10)$ existence question—clearly impractical, but useful as a challenging benchmark for IP and CP solvers.

Acknowledgements.

We thank Yiannis Mourtos for answering many questions concerning his work [39].

References

[1] Appa, G., Magos, D., Mourtos, I.: Searching for mutually orthogonal latin squares via integer and constraint programming. European Journal of Operational Research **173**(2), 519–530 (Sep 2006). https://doi.org/10.1016/j.ejor.2005.01.048

[2] Appa, G., Mourtos, I., Magos, D.: Integrating constraint and integer programming for the orthogonal latin squares problem. In: Lecture Notes in Computer Science, pp. 17–32. Springer Berlin Heidelberg (2002). https://doi.org/10.1007/3-540-46135-3_2

[3] Benadé, J.G., Burger, A.P., van Vuuren, J.H.: The enumeration of k-sets of mutually orthogonal Latin squares. In: Proceedings of the 42th Conference of the Operations Research Society of South Africa, Stellenbosch. pp. 40–49 (2013)

[4] Beth, T.: Eine Bemerkung zur Abschätzung der Anzahl orthogonaler lateinischer Quadrate mittels Siebverfahren. Abh. Math. Sem. Univ. Hamburg **53**, 284–288 (1983). https://doi.org/10.1007/BF02941326

[5] Bose, R.C., Shrikhande, S.S.: On the falsity of Euler's conjecture about the non-existence of two orthogonal Latin squares of order $4t + 2$. Proc. Nat. Acad. Sci. U.S.A. **45**, 734–737 (1959). https://doi.org/10.1073/pnas.45.5.734

[6] Bose, R.C., Shrikhande, S.S., Parker, E.T.: Further results on the construction of mutually orthogonal Latin squares and the falsity of Euler's conjecture. Canadian J. Math. **12**, 189–203 (1960). https://doi.org/10.4153/CJM-1960-016-5

[7] Bright, C., Cheung, K., Stevens, B., Kotsireas, I., Ganesh, V.: A SAT-based resolution of Lam's problem. In: Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence. pp. 3669–3676 (2021), https://ojs.aaai.org/index.php/AAAI/article/view/16483

[8] Bruck, R.H., Ryser, H.J.: The nonexistence of certain finite projective planes. Canad. J. Math. **1**, 88–93 (1949). https://doi.org/10.4153/cjm-1949-009-2

[9] Colbourn, C.J., Dinitz, J.H.: Making the MOLS table. In: Computational and constructive design theory, Math. Appl., vol. 368, pp. 67–134. Kluwer Acad. Publ., Dordrecht (1996). https://doi.org/10.1007/978-1-4757-2497-4_5

[10] Colbourn, C.J., Dinitz, J.H.: Handbook of combinatorial designs. CRC press (2006). https://doi.org/10.1201/9781420010541

[11] Dantzig, G.: Linear programming and extensions. No. 48 in Princeton Landmarks in Mathematics and Physics, Princeton University Press (1963). https://doi.org/10.1515/9781400884179

[12] Dixon, J.D., Mortimer, B.: Permutation groups, Graduate Texts in Mathematics, vol. 163. Springer-Verlag, New York (1996). https://doi.org/10.1007/978-1-4612-0731-3

[13] Dotii, I., Del Val, A., Cebrián, M.: Channeling constraints and value ordering in the quasigroup completion problem. In: Proceedings of the 18th international joint conference on Artificial intelligence. pp. 1372–1373. Citeseer (2003)

[14] Dougherty, S.T.: Mutually orthogonal latin squares. In: Springer Undergraduate Mathematics Series, pp. 75–96. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-56395-0_3

[15] Dubois, O., Dequen, G.: The non-existence of (3, 1, 2)-conjugate orthogonal idempotent latin square of order 10. In: Principles and Practice of Constraint Programming — CP 2001, pp. 108–120. Springer Berlin Heidelberg (2001). https://doi.org/10.1007/3-540-45578-7_8

[16] Elliott, J.R., Gibbons, P.B.: The construction of subsquare free Latin squares by simulated annealing. Australas. J. Combin. **5**, 209–228 (1992)

[17] Euler, L.: Recherches sur un nouvelle espèce de quarrés magiques. Verhandelingen uitgegeven door het zeeuwsch Genootschap der Wetenschappen te Vlissingen **9**, 85–239 (Jan 1782), https://scholarlycommons.pacific.edu/euler-works/530

[18] Gomes, C.P.: Structure, duality, and randomization: Common themes in AI and OR. In: AAAI-00 Proceedings. pp. 1152–1158 (2000), https://www.aaai.org/Papers/AAAI/2000/AAAI00-210.pdf

[19] Gomes, C.P., Regis, R.G., Shmoys, D.B.: An improved approximation algorithm for the partial latin square extension problem. Operations Research Letters **32**(5), 479–484 (2004). https://doi.org/10.1016/j.orl.2003.09.007

[20] Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual, https://www.gurobi.com/documentation/9.1/refman/

[21] Haraguchi, K.: An efficient local search for partial latin square extension problem. In: Integration of AI and OR Techniques in Constraint Programming, pp. 182–198. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-18008-3_13

[22] Hassani, M.: Derangements and applications. J. Integer Seq. **6**(1), Article 03.1.2, 8 (2003)

[23] Hooker, J.N.: Integrated methods for optimization, International series in operations research and management science, vol. 100. Springer (2007). https://doi.org/10.1007/978-1-4614-1900-6

[24] Huang, P., Liu, M., Ge, C., Ma, F., Zhang, J.: Investigating the existence of orthogonal golf designs via satisfiability testing. In: Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation. pp. 203–210 (2019). https://doi.org/10.1145/3326229.3326232

[25] Jin, J., Lv, Y., Ge, C., Ma, F., Zhang, J.: Investigating the existence of costas latin squares via satisfiability testing. In: Theory and Applications of Satisfiability Testing – SAT 2021, pp. 270–279. Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-80223-3_19

[26] Kidd, M.P.: On the existence and enumeration of sets of two or three mutually orthogonal Latin squares with application to sports tournament scheduling. Ph.D. thesis, Stellenbosch University (2012), http://hdl.handle.net/10019.1/20038

[27] Kokkala, J.I., Östergård, P.R.J.: Classification of Graeco-Latin cubes. J. Combin. Des. **23**(12), 509–521 (2015). https://doi.org/10.1002/jcd.21400

[28] Lam, C.W.H., Thiel, L., Swiercz, S.: The nonexistence of finite projective planes of order 10. Canad. J. Math. **41**(6), 1117–1123 (1989). https://doi.org/10.4153/CJM-1989-049-4

[29] Laywine, C.F., Mullen, G.L.: Discrete mathematics using Latin squares. John Wiley & Sons (1998)

[30] Ma, F., Zhang, J.: Finding orthogonal latin squares using finite model searching tools. Science China Information Sciences **56**(3), 1–9 (2013). https://doi.org/10.1007/s11432-011-4343-3

[31] Magos, D.: Tabu search for the planar three-index assignment problem. J. Global Optim. **8**(1), 35–48 (1996). https://doi.org/10.1007/BF00229300

[32] Mariot, L., Picek, S., Jakobovic, D., Leporati, A.: Evolutionary algorithms for the design of orthogonal Latin squares based on cellular automata. In: GECCO'17—Proceedings of the 2017 Genetic and Evolutionary Computation Conference. pp. 306–313. ACM, New York (2017). https://doi.org/10.1145/3071178.3071284

[33] McKay, B.D.: `nauty` graph isomorphism software, available at https://pallini.di.uniroma1.it/ (2021)

[34] McKay, B.D., Meynert, A., Myrvold, W.: Small Latin squares, quasigroups, and loops. J. Combin. Des. **15**(2), 98–119 (2007). https://doi.org/10.1002/jcd.20105

[35] McKay, B.D., Wanless, I.M.: On the number of Latin squares. Ann. Comb. **9**(3), 335–344 (2005). https://doi.org/10.1007/s00026-005-0261-7

[36] Moore, E.H.: Tactical Memoranda I–III. Amer. J. Math. **18**(3–4), 264–303 (1896). https://doi.org/10.2307/2369797

[37] Moura, L.: Polyhedral Methods in Design Theory, pp. 227–254. Springer US, Boston, MA (1996). https://doi.org/10.1007/978-1-4757-2497-4_9

[38] Moura, L.: A polyhedral algorithm for packings and designs. In: Nešetřil, J. (ed.) Algorithms - ESA' 99. pp. 462–475. Springer Berlin Heidelberg, Berlin, Heidelberg (1999). https://doi.org/10.1007/3-540-48481-7_40

[39] Mourtos, I.: Integer and Constraint Programming methods for Mutually Orthogonal Latin Squares. Ph.D. thesis, London School of Economics and Political Science (United Kingdom) (2003), http://etheses.lse.ac.uk/id/eprint/2515

[40] Niskanen, S., Östergøard, P.R.J.: Cliquer user's guide, version 1.0. Tech. Rep. T48, Communications Laboratory, Aalto University, Espoo, Finland (2003)

[41] OEIS Foundation Inc.: The On-Line Encyclopedia of Integer Sequences (2021), http://oeis.org/A002865

[42] OEIS Foundation Inc.: The On-Line Encyclopedia of Integer Sequences (2021), http://oeis.org/A003987

[43] OEIS Foundation Inc.: The On-Line Encyclopedia of Integer Sequences (2021), http://oeis.org/A000315

[44] Perron, L., Furnon, V.: OR-Tools, https://developers.google.com/optimization/

[45] Refalo, P.: Impact-based search strategies for constraint programming. In: Principles and Practice of Constraint Programming – CP 2004, pp. 557–571. Springer Berlin Heidelberg (2004). https://doi.org/10.1007/978-3-540-30201-8_41

[46] Sade, A.: Enumération des Carrés Latins. Application au 7^e Ordre. Conjecture pour les Ordres Supérieurs. Published by the Author, Marseille (1948)

[47] Tarry, G.: Le Problèm des 36 Officiers. Compte Rendu de la Association Française pour l'Avancement des Sciences. **29**(1), 122–123 (1900), https://gallica.bnf.fr/ark:/12148/bpt6k201192t/f3.item

[48] The Sage Developers: SageMath, the Sage Mathematics Software System (Version 9.2) (2021), https://www.sagemath.org

[49] Zaikin, O., Kochemazov, S.: The search for systems of diagonal latin squares using the SAT@home project. International Journal of Open Information Technologies **3**(11), 4–9 (2015)