Improving Integer and Constraint Programming for Graeco-Latin Squares

Noah Rubin*, Curtis Bright†*, Kevin Cheung*, Brett Stevens*
*Carleton University
†University of Windsor

***Abstract*—We use integer programming (IP) and constraint programming (CP) to search for graeco-latin squares. 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 graeco-latin squares (or prove their nonexistence) in all orders up to and including eleven.**

I. Introduction

A *latin square of order* n is an $n \times n$ array, X, 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 X is denoted X_{ij}. Two latin squares of the same order, X and Y, are said to be *orthogonal* if there is a unique solution $X_{ij} = a$, $Y_{ij} = b$ for every pair of $a, b \in \{0, 1, \ldots, n-1\}$. A set of latin squares of order n is called a set of *mutually orthogonal latin squares* if all squares are pairwise orthogonal. A pair of orthogonal latin squares are also known as a *graeco-latin square*. 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 [6], [10], [25].

There is a long history of using "automated reasoning" tools like constraint and integer programming solvers to search for and construct latin squares [14]. 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 [17]. There has been much work developing improved solvers for such problems [9], [11], [37]. 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 graeco-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 graeco-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 graeco-latin squares for orders $n \leq 11$ with this improved encoding compared to $n \leq 8$ with the traditional encoding and symmetry breaking. 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 search by an exponential factor in the order 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 graeco-latin squares for orders $n \leq 10$ 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.

Lastly, we give timings for each method in Section IV. 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 [19].

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 orthogonal latin squares and develop sophisticated techniques for combining solvers in ways that exploit each solver's strengths [1], [2].

III. Models

In this section we describe the models that we use when searching for graeco-latin squares 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 graeco-latin square problem is to express it as a pure binary linear program and use integer programming (IP) solvers to generate solutions [7, §26.3.IV]. Our IP model for finding a graeco-latin square pair X and Y contains the n^4 binary variables

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

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

We encode the latin and orthogonality constraints as $\binom{4}{2} = 6$ sets of n^2 equalities grouped by which two subscripts of x_{ijkl} are fixed. For example, fixing i and j leads to the constraint $\sum_{0 \le k,l < n} x_{ijkl} = 1$ which says that cell (i, j) only contains a single value. Fixing k and l leads to the constraint $\sum_{0 \le i,j < n} x_{ijkl} = 1$ which when taken over all $0 \le k, l < n$ says that X and Y are orthogonal.

B. Constraint Programming Model

A CP solver allows a more natural formulation of the graeco-latin square problem using the $2n^2$ integer-valued variables X_{ij} and Y_{ij} (for $0 \le i, j < n$) directly encoding the values appearing in cell (i, j) of squares X and Y. The squares 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

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

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. The variables Z_{ij} may equivalently be defined via the modulo and division constraints $X_{ij} = Z_{ij} \bmod n$ and $Y_{ij} = \lfloor Z_{ij}/n \rfloor$ and we call this the CP-moddiv 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,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 [25, 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

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

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 [36] 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.

C. Symmetry Breaking

There are a large number of symmetries in the graeco-latin square problem. If X and Y form a graeco-latin square then X and Y can be permuted, the rows of X and Y can be permuted simultaneously, the columns of X and Y can be permuted simultaneously, the symbol sets within X and Y can be permuted independently, and the squares may be replaced with their transposes. All of these operations preserve the latin properties and orthogonality of the squares [6]. This group of possible symmetries has order $4(n!)^4$ 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 X to be in lexicographic order which eliminates permutations of the rows. This reduces the possible symmetries to $4 \cdot n!$. 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 $(n-1)!/e$ [18].

Appa *et al.* [1], [2] further show that every possible solution is isomorphic to one satisfying $Y_{i0} \neq i$ 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 $\{\rho \in P(n) : \rho(0) = 0,\ \rho(i) \neq i,\ \rho(i) \leq i+1\}$, and $a(n) = |A(n)|$.

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

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. By exploiting the disjoint cycle structure of these permutations, we have been able to reduce this number to $b(n)$, 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 [35]

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

This growth rate is exponentially slower than the Fibonacci numbers.

We say that a graeco-latin square (X, Y) of order n is in *standard form* if the first rows of X and Y 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 $\{1, 2, \ldots, n-1\}$ defined by $\rho(i) = Y_{i0}$. The *ordered cycle* of Y is the representation of ρ as a product of disjoint cycles

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

where each cycle is the lexicographic least cyclic shift and the cycles are in lexicographic order [8]. The *ordered cycle type* of Y is $(l_0, l_1, \ldots, l_c)$. The following theorem captures our symmetry breaking method.

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

Proofs of Proposition 1 and Theorem 1 can be found online [38].

Model	5	6	7	8	9	10
IP	0.1	Timeout	3.2	6.4	344.5	3,046.4
CP-linear	0.0	Timeout	7.4	1,721.7	50,753.7	Timeout
CP-moddiv	0.4	Timeout	7.7	509.9	1,672.5	Timeout
CP-index	0.0	Timeout	3.5	82.4	1,552.1	11,971.3

TABLE I
BASELINE TIMINGS (IN SECONDS) FOR ORDERS $5 \leq n \leq 10$ WITH NO SYMMETRY BREAKING AND A TIMEOUT OF 60,000 SECONDS.

IV. RESULTS

In this section we provide running times for our implementations. 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 [16] and the CP models were solved using OR-Tools version 8 [36] whose CP solver uses lazy clause generation in tandem with a conflict-driven SAT solver [40]. 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.

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. The instances in order six are infeasible and required symmetry breaking to solve; in every other order $5 \leq n \leq 11$ the solver either explicitly found a graeco-latin square or timed out after 60,000 seconds. In Table I we compare the IP model along with the three CP models—one with linear constraints encoded directly (CP-linear), one with linear constraints encoded via modulo and division constraints (CP-moddiv), and one with indexing constraints (CP-index). The timings show that the encoding based on indexing constraints generally outperformed the traditional encoding based on linear constraints.

One reason why the CP-linear model is particularly slow may be due to OR-Tools not achieving a strong level of local consistency on its linear constraints during constraint propagation. In order to fairly compare the traditional "linear" orthogonality encoding with our "indexing" orthogonality encoding we implemented the CP-moddiv model using indexing constraints—this way the level of propagation achieved in the orthogonality constraints of the CP-moddiv and CP-index models should be the same. In detail, the orthogonality variables $Z_{ij} = X_{ij} + nY_{ij}$ in the CP-moddiv model were specified through the indexing constraints

$$X_{ij} = \left[\, k \bmod n : 0 \leq k < n^2 \,\right][Z_{ij}], \text{ and}$$
$$Y_{ij} = \left[\, \lfloor k/n \rfloor : 0 \leq k < n^2 \,\right][Z_{ij}]$$

for $0 \leq i, j < n$. The fact that the CP-index model tends to outperform the CP-moddiv model we believe is evidence

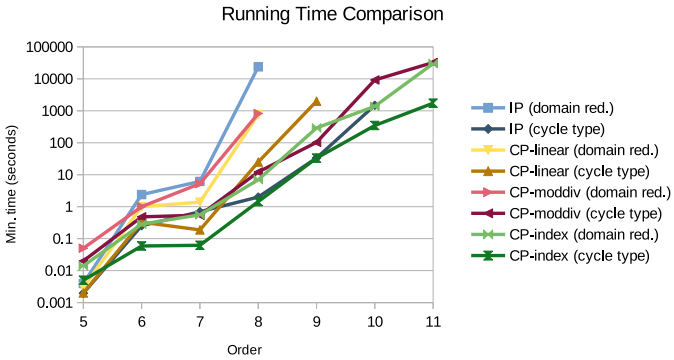

Fig. 2. A comparison of the running times for the various models and symmetry breaking methods that we considered in the orders $5 \leq n \leq 11$.

for the effectiveness of the orthogonality encoding based on realizing the latin square $Z = X^{-1}Y$.

Figure 2 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. 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. 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.

V. Related Work and History

Graeco-latin squares of order 4 were known to medieval Muslim mathematicians. In 1700, the Korean mathematician Choi Seok-jeong presented a graeco-latin square of order 9 and reported being unable to find a pair of order 10 [6]. Euler could construct graeco-latin squares for all odd n and for n divisible by 4 [13]. He knew that graeco-latin squares of order 2 did not exist 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 graeco-latin squares of order n exist if and only if $n \not\equiv 2 \pmod 4$.

Many mathematical methods for constructing sets of mutually orthogonal latin squares (MOLS) exist. These range from algebraic to the recursive [6]. 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 nine MOLS of order ten was an exhaustive backtracking search aided by powerful theorems from coding theory and permutation groups to eliminate finding structures isomorphic to those already found [24]. McKay, Meynert, and Myrvold use the orderly generation method to enumerate all non-isomorphic latin squares of orders up to nine [30]. They use the library `nauty` [29] to compute automorphisms and canonical representatives in each class and enumerate the different numbers of equivalence classes of these squares. McKay and Wanless are able to enumerate all latin squares of order 11 [31] 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 [31], [39]. Niskanen and Östergård's clique finding software, `cliquer` has been successfully used to find mutually orthogonal latin cubes [23], [34]. Kidd [22] 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 nine MOLS of order ten was confirmed by Bright *et al.* [4] by producing nonexistence proof certificates.

Colbourn and Dinitz [5] implemented many of the combinatorial constructions to aid generating the tables given in the Handbook of Combinatorial Designs [6]; strictly speaking, these computational methods are not metaheuristics but neither are they exhaustive searches. Elliott and Gibbons [12] use the *simulated annealing* metaheuristic to construct latin squares of orders up to 18 which contain no subsquares. Magos [27] has used the Tabu metaheuristic to construct latin squares. Mariot *et al.* [28] 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 [32], [33] uses integer programming (IP) to search for certain combinatorial designs that are related to latin squares. Huang *et al.* [20] use constraint programming (CP) to search for orthogonal golf designs. FeiFei and Jian [26] use a first-order logic model generator to search for orthogonal latin squares while Zaikin and Kochemazov [41] use a propositional logic solver (i.e., SAT solver) to search for orthogonal diagonal latin squares and Jin *et al.* [21] use a SAT solver to search for Costas latin squares. Gomes, Regis, and Shmoys [15] 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 three MOLS of order ten if such a triple exists.

VI. Conclusion

In this paper we use integer and constraint programming solvers in the search for graeco-latin squares. We demonstrate that modern state-of-the-art solvers are able to find graeco-latin squares (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.

Acknowledgments: We thank the reviewers for their helpful comments—in particular, raising the point about the level of local consistency achieved during propagation.

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] 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

[5] 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

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

[7] 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

[8] 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

[9] 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)

[10] 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

[11] 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

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

[13] 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

[14] 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

[15] 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

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

[17] 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

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

[19] 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

[20] 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

[21] 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

[22] 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

[23] 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

[24] 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

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

[26] 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

[27] 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

[28] 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

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

[30] 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

[31] 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

[32] 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

[33] 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

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

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

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

[37] 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

[38] Rubin, N., Bright, C., Cheung, K.K.H., Stevens, B.: Integer and constraint programming revisited for mutually orthogonal latin squares. arXiv:2103.11018 (2021), https://arxiv.org/abs/2103.11018

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

[40] Stuckey, P.J.: Lazy clause generation: Combining the power of SAT and CP (and MIP?) solving. In: Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pp. 5–9. Springer (2010). https://doi.org/10.1007/978-3-642-13520-0_3

[41] 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)