Integer and Constraint Programming Revisited for Mutually Orthogonal Latin Squares (Student Abstract)

Noah Rubin, Curtis Bright, Brett Stevens, Kevin Cheung

noahrubin@cmail.carleton.ca, cbright@uwindsor.ca, brett@math.carleton.ca, kevincheung@cunet.carleton.ca

Abstract

We use integer programming (IP) and constraint programming (CP) to search for sets of mutually orthogonal latin squares (MOLS). We build upon the work of Appa *et al.* in their paper "Searching for Mutually Orthogonal Latin Squares via Integer and Constraint Programming"[1] by formulating an extended symmetry breaking method and providing an alternative CP encoding which performs much better in practice.

Orthogonal Latin Squares

A *latin square of order* n is an $n \times n$ array, L, of symbols $\{0, 1, \dots, n-1\}$ in which each symbol appears exactly once in each row and column. 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 $a, b \in \{0, 1, \dots, n-1\}$. 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 kMOLS(n).

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

Figure 1: An example of 3MOLS(4).

Symmetry Breaking

Solutions to the kMOLS(n) problem are isomorphic to many other solutions, so we impose constraints on the domains of cells to reduce symmetries. Let X, Y be a set of 2MOLS(n), we impose the following:

1. Fix first row of X and Y in lex order to prevent permutations of symbols within squares.
2. Fix first column of X in lex order to prevent permutations of rows across all squares.
3. Fix first column of Y to be one of a_n tuples with distinct cycle types.

The last strategy follows from Theorem 1 in our supplementary submission material, where a detailed proof is also provided. The value a_n is given in entry A002865 of the Online Encyclopedia of Integer Sequences[2] and has growth rate $e^{O(\sqrt{n})}$.

IP Model

Our IP model for 2MOLS(n) uses 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 $i, j, k, l \in \{0, 1, \dots, n-1\}$. The Latin and orthogonality constraints are expressed as

$$\sum_{0 \le k,l < n} x_{ijkl} = 1 \; \forall i,j \quad \text{1 value per cell}$$

$$\sum_{0 \le j,l < n} x_{ijkl} = 1 \; \forall i,k \quad \text{Latin rows of } X$$

$$\sum_{0 \le j,k < n} x_{ijkl} = 1 \; \forall i,l \quad \text{Latin rows of } Y$$

$$\sum_{0 \le i,l < n} x_{ijkl} = 1 \; \forall j,k \quad \text{Latin columns of } X$$

$$\sum_{0 \le i,k < n} x_{ijkl} = 1 \; \forall j,l \quad \text{Latin columns of } Y$$

$$\sum_{0 \le i,j < n} x_{ijkl} = 1 \; \forall k,l \quad X, Y \text{ orthogonal}$$

CP Model

Our CP model for 2MOLS(n) uses $2n^2$ integer variables

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

Where $i, j, X_{ij}, Y_{ij}, Z_{ij} \in \{0, 1, \dots, n-1\}$ and the ith row of Z is row i of Y applied to the inverse of row i of X. X, Y are orthogonal if and only if Z is Latin [3, Theorem 6.6], so we impose orthogonality by ensuring X, Y and Z are Latin squares.

$$\text{AllDifferent}(X_{ij}\ \forall j), \text{AllDifferent}(X_{ij}\ \forall i),$$
$$\text{AllDifferent}(Y_{ij}\ \forall j), \text{AllDifferent}(Y_{ij}\ \forall i),$$
$$\text{AllDifferent}(Z_{ij}\ \forall j), \text{AllDifferent}(Z_{ij}\ \forall i)$$

To make $Y = XZ$ the (i,j)th entry of Y is set to the (i, X_{ij})th entry of Z using "element indexing" constraints $Z_i[X_{ij}] = Y_{ij}$ where Z_i is the vector of variables corresponding to row i of Z.

Results

We ran trials on a computer with an Intel i9 9900k processor and 32GB of memory. Trials were allocated 1 core each and timeout was set at 60,000s. The implementations of our programs were done in Microsoft Visual C++ and later modified to run in Linux. We used Gurobi[5] as our IP solver and Google OR-Tools[4] as our CP solver. Each are highly competitive in their class, and are free to use for students. The CP-linear model given by Appa *et al.* [1] was originally used, which imposed orthogonality by defining $Z_{ij} :- X_{ij} + nY_{ij}$ and AllDifferent$(Z_{ij}) \forall i, j$. We later revised this to our CP-index model, which outperformed all other models even with no symmetry breaking.

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	8.0	1,967.1	58,637.8	Timeout
CP-index	0.0	Timeout	7.8	36.3	378.7	214.8

Table 1: Timings in seconds for orders $5 \le n \le 10$ with no symmetry breaking.

Imposing all of the symmetry breaking strategies gave us significant improvements in the running time of all models. Symmetry breaking strategy 3 is an extension of the strategy used by Appa *et al.* [1], who show that any solution to kMOLS(n) is isomorphic to one where $Y_{10} = 2$, $Y_{i0} \neq i$ and $Y_{i0} \le i + 1$ for $1 \le i < n$, with the total number of fixings given by the $(n-2)^{\text{th}}$ Fibonacci number. Imposing symmetry breaking strategy 3 reduces this number to a_n.

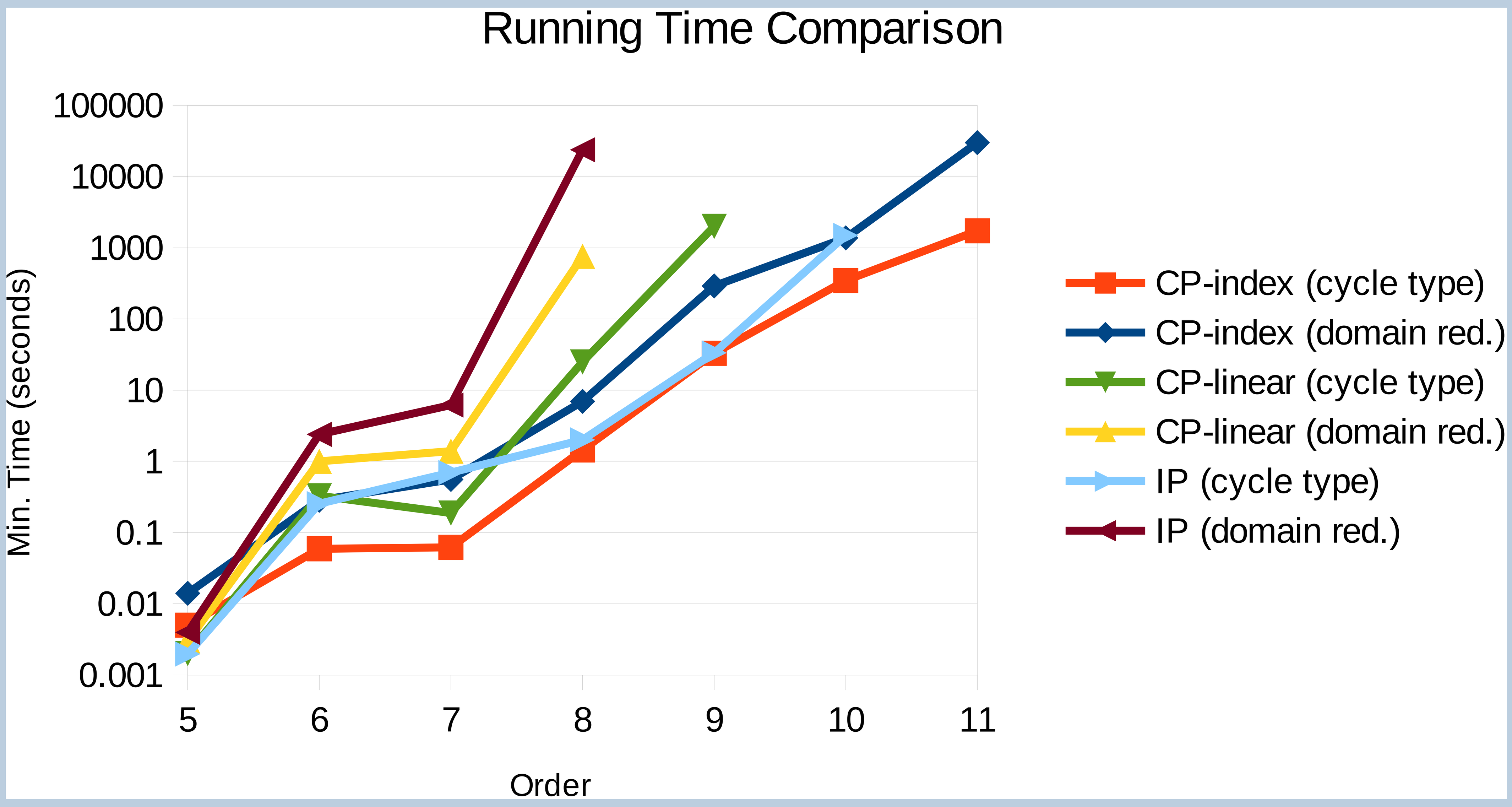

Figure 2: Running times of our models and symmetry breaking methods for $5 \le n \le 11$.

More detailed timings, proofs and complete implementations of our programs can be found at https://github.com/noahrubin333/CP-IP.

References

[1] G. Appa, D. Magos, and I. Mourtos. Searching for mutually orthogonal latin squares via integer and constraint programming. *European Journal of Operational Research*, 173(2):519–530, September 2006.

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

[3] Charles F. Laywine and Gary L. Mullen. *Discrete mathematics using Latin squares.* John Wiley & Sons, 1998.

[4] Laurent Perron and Vincent Furnon. OR-Tools, 2019. https://developers.google.com/optimization/.

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