A constraint programming formulation for finding a pair of MOLs

Barbara Smith

July 14, 2010

Suppose we are required to find a pair of $n \times n$ MOLs. In what follows, the pair of squares is sometimes viewed as a single $n \times n$ matrix with each cell containing a single number, formed from the pair of numbers in the original pair of Latin squares. I will call this the pairs matrix. I also switch between viewing the elements in the pairs matrix as a pair of numbers, each in the range $0, \ldots, n-1$ and a single number in the range $0, \ldots, n^2-1$.

Let x_{ij} be the number in row i and column j of square 1, and y_{ij} be the number in row i and column j of square 2, for $0 \leq i < n, 0 \leq j < n$. The domain of each variable is $\{0, ..., n-1\}$.

There are all-different constraints on these variables. For each row, the numbers in the row are different:

$$\text{allDiff}(x_{i0}, x_{i1}, \ldots, x_{i,n-1}) \quad \text{for all } 0 \leq i < n$$

For each column, the numbers in the column are different:

$$\text{allDiff}(x_{0j}, x_{1j}, \ldots, x_{n-1,j}) \quad \text{for all } 0 \leq j < n$$

and similar constraints on the rows and columns of the second square.

Let s_{ij} represent the number pair in row i and column j of the pair of squares. The domain of s_{ij} is $\{0, ..., n^2-1\}$

The constraints linking these variables are:

$$s_{ij} = n \times x_{ij} + y_{ij} \quad \text{for all} \;\; i, j$$

These variables must all have different values, but it will turn out that we don't need to express that as a constraint.

Dual variables: Let s'_{ij} represent the cell of the pairs matrix that contains the number pair (i, j), or equivalently the number $ni + j$.

I introduce new variables x'_{ij}, y'_{ij} defined by:

$$s'_{ij} = n \times x'_{ij} + y'_{ij} \quad \text{for all} \;\; i, j$$

i.e. $x'_{ij} = k$ iff the pair of numbers (i, j) occurs in row k, $y'_{ij} = l$ iff the pair of numbers (i, j) occurs in row l.

Consider the variables $x'_{i0}, x'_{i1}, \ldots, x'_{i,n-1}$. These represent the rows in which the pairs $(i, 0), (i, 1), \ldots, (i, n-1)$ occur. If any two of these pairs occur in the same row, then that row has two occurrences of the value i in the first square, which violates the Latin square condition. hence we have the constraint:

$$\text{allDiff}(x'_{i0}, x'_{i1}, \ldots, x'_{i,n-1}) \quad \text{for all } 0 \le i < n$$

Similarly, we have the constraint:

$$\text{allDiff}(x'_{0j}, x'_{1j}, \ldots, x'_{n-1,j}) \quad \text{for all } 0 \le j < n$$

and the equivalent constraints on the y'_{ij} variables.

Finally, the s and s' variables are duals, and can be linked by the channelling constraints:

$$s_{ij} = (k, l) \text{ iff } \; s'_{kl} = (i, j)$$

Because of these channelling constraints, we don't need the constraints that the s_{ij} variables (and also the s'_{kl} variables) are all different.

I also have the straightforward symmetry constraints, that the top row in both squares and the first column in the first square are $0, 1, ..., n-1$. Also the pair in the second row, 1st column is $(1, 2)$.

Search strategy: I assign the x and y variables alternately, i.e. $x_{00}, y_{00}, x_{01}, y_{01}, \ldots$ Smallest-domain-first is not as good as good as keeping to this order.

Performance: with this model in ILOG Solver 6.6, proving that there is no 6×6 square takes 5529 backtracks and just over 1 second on a 1.6 GHz laptop.

Notes:

- it is important to maintain Generalized Arc Consistency on the allDifferent constraints. Solver allows the user to choose whether to do that or to maintain a lower level of consistency.
- In Solver, the channelling constraints can be expressed using a single IloInverse constraint that links two arrays of variables and is equivalent to the set of individual channelling constraints listed earlier, in the sense that it does the same propagation but is faster.
- It might be possible to add more symmetry constraints, but I am still looking into this.