Decision Intelligence

Spatially Balanced Latin Squares using Constraint Programming

Réalisé par :
Ala Eddine Ben Mimoun

Encadré par :
REGIN Jean-Charles

Master Informatique
Université Côte d'Azur
Premier semestre de l'année 2025/2026

Contents

1 Introduction

The Spatially Balanced Latin Square (SBLS) problem originates from experimental design, notably in agriculture. When testing several treatments (fertilizers), spatial proximity between plots may introduce bias due to interactions between neighboring plots.

A classical Latin Square ensures that each treatment appears exactly once in each row and column. However, it does not control spatial proximity between treatments. The SBLS problem extends the Latin Square by imposing an additional constraint: the spatial distribution of each pair of treatments must be balanced.

This project aims to model and solve the SBLS problem using Constraint Programming (CP) with the **Choco Solver**. We study the impact of spatial constraints on the computational complexity and compare the SBLS model with a simpler baseline model.

2 Problem Definition

Given an integer n, we consider an $n \times n$ grid. Each cell contains a value in $\{0, \ldots, n-1\}$ representing a fertilizer.

The goal is to:

- Assign fertilizers to grid cells
- Respect Latin Square constraints
- Ensure spatial balance between all pairs of fertilizers

Spatial balance is defined as follows:

> For every pair of distinct fertilizers (c_1, c_2), the sum of Manhattan distances between all occurrences of c_1 and c_2 must be identical for all pairs.

3 Constraint Programming Model

3.1 Variables

We define a matrix of decision variables:

$$grid[i][j] \in \{0, \ldots, n-1\}$$

Each variable represents the fertilizer placed at row i, column j.

3.2 Constraints Used

3.2.1 Latin Square Constraints

Row constraints Each fertilizer must appear exactly once in each row:

$$AllDifferent(grid[i][0..n-1])$$

Column constraints Each fertilizer must appear exactly once in each column:

$$AllDifferent(grid[0..n-1][j])$$

3.2.2 Symmetry Breaking

Latin Squares contain many symmetric solutions. To reduce the search space, we fix the first row:

$$grid[0][j] = j$$

This does not remove any valid solutions up to isomorphism and significantly improves performance.

3.2.3 Spatial Balance Constraints

For each pair of fertilizers (c_1, c_2), we compute:

$$\sum_{(i_1,j_1),(i_2,j_2)} d((i_1, j_1), (i_2, j_2))$$

where:

$$d = |i_1 - i_2| + |j_1 - j_2|$$

All such sums are constrained to be equal.

4 Filtering Algorithms

4.1 AllDifferent Constraint

The `AllDifferent` constraint in Choco uses:

- **Regin's algorithm**
- Based on bipartite matching
- Enforces Generalized Arc Consistency (GAC)

This is essential for pruning inconsistent values early.

4.2 Arithmetic Constraints

Constraints such as equality and sums use:

- Bound consistency
- Domain filtering through propagation

4.3 Conditional Constraints

The spatial constraints rely on `ifThenElse`. These constraints are propagated using logical reification and standard arithmetic filtering.

5 Search Strategy

5.1 Variable Selection

We use:

`minDomLBSearch`

which selects:

- The variable with the smallest domain
- Breaks ties using the lowest bound

This follows the **fail-first principle**.

5.2 Value Selection

Values are assigned in increasing order. This deterministic strategy ensures reproducibility.

6 Baseline Method

As a comparison, we define a **Simple Latin Square** model:

- Same variables
- Only row and column `AllDifferent` constraints
- No spatial balance constraints

This baseline highlights the computational cost introduced by spatial balancing.

7 Experimental Results

7.1 Machine Information

- Processor: AMD Ryzen 5 5600H
- Cores: 6 (12 threads)
- RAM: 16 GB
- OS: Linux / Windows
- Solver: Choco Solver 4.x

7.2 Results

The following table reports the experimental results obtained on our machine. A timeout was set to 60 seconds. For SBLS, unsatisfiability is often detected during propagation, before any search node is explored.

n	Method	Time (ms)	Nodes
2	Simple LS	2	2
2	SBLS	1	0 (UNSAT)
3	Simple LS	1	4
3	SBLS	7	0 (UNSAT)
4	Simple LS	2	7
4	SBLS	33	0 (UNSAT)
5	Simple LS	2	11
5	SBLS	98	0 (UNSAT)
6	Simple LS	2	19
6	SBLS	290	0 (UNSAT)
7	Simple LS	2	26
7	SBLS	811	0 (UNSAT)
8	Simple LS	3	36
8	SBLS	1594	0 (UNSAT)
9	Simple LS	2	50
9	SBLS	3282	0 (UNSAT)

7.3 Discussion

The experimental results highlight a fundamental difference between the two models.

The Simple Latin Square is easily solvable for all tested values of n, with very low solving time and a small number of search nodes.

In contrast, the SBLS model is proven unsatisfiable for all tested values of n. This unsatisfiability is detected by constraint propagation alone, before any search node is explored. As a consequence, the number of explored nodes is equal to zero.

This behavior shows the strength of the filtering algorithms used, particularly the `AllDifferent` constraint with generalized arc consistency and the interaction between arithmetic and reified constraints.

For larger values of n, the number of constraints grows rapidly (in $O(n^4)$), leading to memory exhaustion during model construction. This illustrates the practical limits of naive SBLS modeling and confirms the intrinsic difficulty of the problem.

8 Conclusion

In this project, we modeled the Spatially Balanced Latin Square problem using Constraint Programming with Choco Solver.

We demonstrated that:

- SBLS can be naturally expressed using CP
- Spatial constraints drastically increase complexity
- Symmetry breaking is essential
- SBLS is practically limited to small values of n

Future work could explore:

- Stronger global constraints
- Decomposition techniques
- Metaheuristics or hybrid approaches