Rapport de Projet : Spatially Balanced Latin Square (SBLS)

DA COSTA Tom

Projet SBLS - M1 2025

Janvier 2026

1 Introduction

Le problème du Carré Latin Spatialement Équilibré (SBLS) trouve son origine dans l'aménagement de parcelles agricoles. L'objectif est de tester n types d'engrais, représentés par des couleurs, sur des parcelles disposées en grille. Une disposition classique peut introduire des biais dus à la proximité d'éléments extérieurs (forêt, route) ou à des caractéristiques du sol (source d'eau).

En divisant le champ en $n \times n$ parcelles, si deux engrais sont fréquemment voisins, des échanges entre eux peuvent fausser les résultats. Le SBLS vise à éliminer ce biais : dans le carré latin, la somme des distances entre chaque paire de couleurs doit être identique. L'objectif de ce projet est de modéliser ce problème en programmation par contraintes (CP) avec Choco Solver.

2 Modélisation en Programmation par Contraintes

2.1 Variables de décision

Nous utilisons une matrice de variables entières $X[n][n]$ où chaque cellule $x_{i,j}$ peut prendre une valeur comprise entre 0 et $n-1$, représentant les n engrais disponibles.

2.2 Contraintes utilisées

Le modèle repose sur trois contraintes :

— **Contraintes de Carré Latin** : Pour garantir que chaque engrais apparaît exactement une fois par ligne et par colonne, nous utilisons la contrainte globale `allDifferent` sur chaque ligne et chaque colonne.

— **Contrainte d'équilibre spatial** : Pour chaque paire de couleurs (a, b), nous imposons que la somme des distances de Manhattan entre toutes les positions occupées par a et b soit égale à une constante K ,K étant la somme des distances entre toutes les occurences de deux couleurs distinctes.

— **Casse de symétrie** : La première ligne et la première colonne sont fixées à l'ordre naturel $(0, 1, \ldots, n-1)$ pour réduire l'espace de recherche.

2.3 Algorithmes de filtrage

Pour optimiser la résolution dans Choco Solver, j'ai configuré :

— **allDifferent** : Filtrage `AC` (Arc Consistency), pour une réduction maximale des domaines.

— **Réification** ($x == v$) : Cette contrainte fait le lien entre une case du carré et un booléen. Si le solveur décide qu'une case vaut v, le booléen passe à 1. Si le booléen passe à 0, le solveur retire v des possibilités de la case. Le filtrage est instantané dans les deux sens.

— **Contraintes AND** : Pour calculer l'équilibre spatial, on doit savoir si deux engrais sont à des positions précises en même temps. On utilise des clauses booléennes. Si le résultat du AND est "vrai", Choco force immédiatement les deux variables d'entrée à être "vraies".

— **Contraintes arithmétiques** : Les sommes pondérées pour le calcul de K utilisent un filtrage `BC` (Bound Consistency). Il calcule la somme minimale et maximale possible avec les valeurs restantes. Si le total dépasse K ou ne peut plus l'atteindre, il retire les valeurs qui posent problème.

2.4 Justification des choix de modélisation

— **Distance de Manhattan** : Ce choix est le plus pertinent pour une grille de parcelles. Elle représente le déplacement réel (horizontal et vertical) entre deux points, calculé par la formule $d(p, q) = |r_1 - r_2| + |c_1 - c_2|$.

— **Encodage booléen** : Comme les variables de base représentent des couleurs, nous avons utilisé des variables booléennes pour isoler chaque fertilisant. Cela permet de n'activer les distances dans le calcul de la somme que lorsque les deux couleurs cibles sont présentes sur les cases comparées.

— **Pourquoi DomOverWDeg et pas une autre stratégie ?** : Contrairement à une stratégie statique qui remplit les cases toujours dans le même ordre (comme `inputOrder`), `DomOverWDeg` est une stratégie dynamique. Elle donne une sorte de "score de difficulté" aux cases en fonction des échecs passés du solveur. Elle choisit donc de remplir en priorité les zones les plus dures du carré. Pour le SBLS, c'est indispensable car les contraintes de distance sont très restrictives. En réglant les problèmes les plus difficiles dès le début, le solveur gagne énormément de temps en évitant d'explorer des branches qui n'ont aucune solution.

— **Bris de symétrie** : En fixant la première ligne et la première colonne, on empêche le solveur de recalculer des solutions identiques par simple rotation ou permutation des couleurs. Cela réduit drastiquement le nombre de feuilles à explorer dans l'arbre de recherche.

3 Stratégies de recherche

Les stratégies choisies pour maximiser l'ordre n sont :

— **Variable Strategy** : `DomOverWDeg`, qui sélectionne les variables les plus "difficiles" en fonction des échecs passés.

— **Value Strategy** : `IntDomainMin`, sélectionnant la plus petite valeur disponible.

4 Expérimentations et Résultats

La méthode SBLS est comparée à une méthode "simple" (Carré latin standard sans équilibre spatial).

4.1 Environnement de test

— **CPU** : 13th Gen Intel(R) Core(TM) i7-1370P
— **RAM** : 32 GB
— **OS** : Windows 11 10.0

4.2 Tableau comparatif

n	Méthode	Temps (s)	Nœuds	Solutions
2	Simple	0.002	1	1
2	SBLS	0.001	1	1
3	Simple	0.000	1	1
3	SBLS	0.012	1	1
4	Simple	0.002	7	4
4	SBLS	0.010	7	4
5	Simple	0.012	111	56
5	SBLS	0.085	111	56
6	Simple	0.307	18815	9408
6	SBLS	8.559	18873	9408
7	Simple	283,791	33884159	16942080
7	SBLS	inf	Time Out	Time Out

TABLE 1 – Comparaison des performances entre le modèle simple et le modèle SBLS.

5 Analyse des résultats et limites du modèle

5.1 Pourquoi le temps de calcul explose pour n = 7 ?

D'après nos tests, le passage à n = 7 marque une limite matérielle et algorithmique pour plusieurs raisons :

1. **Complexité combinatoire** : Le nombre de carrés latins possibles croît de manière factorielle avec n. L'espace de recherche pour n=7 est massivement plus grand que pour n=6.
2. **Volume des contraintes** : Notre modélisation de l'équilibre spatial compare chaque case avec toutes les autres cases pour chaque paire de couleurs. Le nombre de variables et de contraintes générées augmente ainsi en n^4, ce qui sature la mémoire et le processeur.

3. **Saturation du filtrage** : À chaque nœud de l'arbre, le solveur doit propager les domaines de milliers de variables liées entre elles. Pour n=7, le temps passé par Choco à nettoyer les domaines (filtrage AC et BC) devient plus long que la recherche elle-même.

6 Conclusion

L'ajout des contraintes de balance spatiale complexifie la résolution, mais l'utilisation de filtrages robustes (AC) permet d'obtenir des solutions valides. Le modèle permet de trouver l'ordre n maximal pour les besoins agricoles identifiés.