

# Résolution de Sudoku

*Graphes, Complexité et Combinatoire*

---

AUBOURG Thomas, DEMEUDÉ Edgar et VU Anh Duy

# Plan

I - Benchmark

---

II - Méthodes complètes

---

III - Méthodes incomplètes



# I - Benchmark

# Objectifs : Avoir un jeu de données varié et calibré pour tester les limites des solveurs

## Geocode

Benchmark académique standard

## Génération

Instances générées par Z3.



# Instances récupérées

**80 instances de Sudoku au format .dzn**  
*(de dimensions différentes: 9x9, 16x16, and 25x25)*

**Issu de la librairie Gecode / CSPLib**

**Académique standard**  
*(utilisé dans MiniZinc Challenge et par Hakan Kjellerstrand)*

# Instances générées

## 1. Les Contraintes (Modélisation SMT)

- Valeurs possibles (Domaine)
- Unicité horizontale (Lignes)
- Unicité verticale (Colonnes)
- Unicité par "Carré" (Sous-grilles)

## 2. L'Oracle de Génération : Z3

- Rôle : moteur génératif
- Approche Déclarative
- Maîtrise de l'Aléatoire et de la Diversité

*Microsoft Research → Puissance de calcul et algorithmes avancés*

*Z3 est un solveur de satisfaisabilité formelle.*

*Exploration Aléatoire Contrôlée*

Echec des 36x36

Densité	Sudoku 16x16	Sudoku 25x25	Sudoku 36x36
20%	3.2 s	118 s	Timeout (> 1h)
30%	4.1 s	132 s	Timeout (> 1h)
40%	3.8 s	124 s	Timeout (> 1h)
50%	4.5 s	145 s	Timeout (> 1h)
60%	3.5 s	112 s	Timeout (> 1h)
70%	3.9 s	129 s	Timeout (> 1h)
80%	4.2 s	138 s	Timeout (> 1h)
Moyenne	~ 3.9 s	~ 128 s (2min 08)	Échec

## 3. Densité

- Ratio cases remplies / cases vides (de 0.2 à 0.8)
- Méthode "Punching Holes"
- 2 instances par couple (taille, densité)

# RÉSOLUTION COMPLÈTE DE SUDOKU AVEC CHOCO SOLVER

Approches Complètes et Optimisation par Contraintes



# MODÉLISATION ET CONTRAINTES


Contrainte AllDifferent

Propagation Active (.post())

Index = blockRow\*n + i

--	--	--	--	--	--	--	--	--

Tableau 1D (Flatten)

# MOTEUR DE CONFIGURATION

Strategy de variables	Heuristique de valeur	Niveau de consistence	Type de redémarrage
InputOrder ( Default)	MIN ( Default)	DepthFirstSearch ( Default)	NO RESTART ( Default)
DomOverWDeg	MAX	AC	LUBY(10, 100, 200, 500)
MinDomSize	MIDDLE		GEOMETRIQUE
	RANDOM		

# LE DÉFI & LE BENCHMARK

4	1	2	3	9	6	7	8	1
1	2	3	6	9	7	4	9	4
2	4	7	1	8	9	3	6	5
6	6	5	8	4	1	2	3	9
4	7	1	6	1	2	8	2	5
1	2	8	3	9	5	7	4	7
9	8	3	6	7	4	2	1	8
7	4	2	7	9	1	4	6	1
3	7	5	5	1	4	9	7	6

# Easy 9x9

## 9<sup>81</sup> états

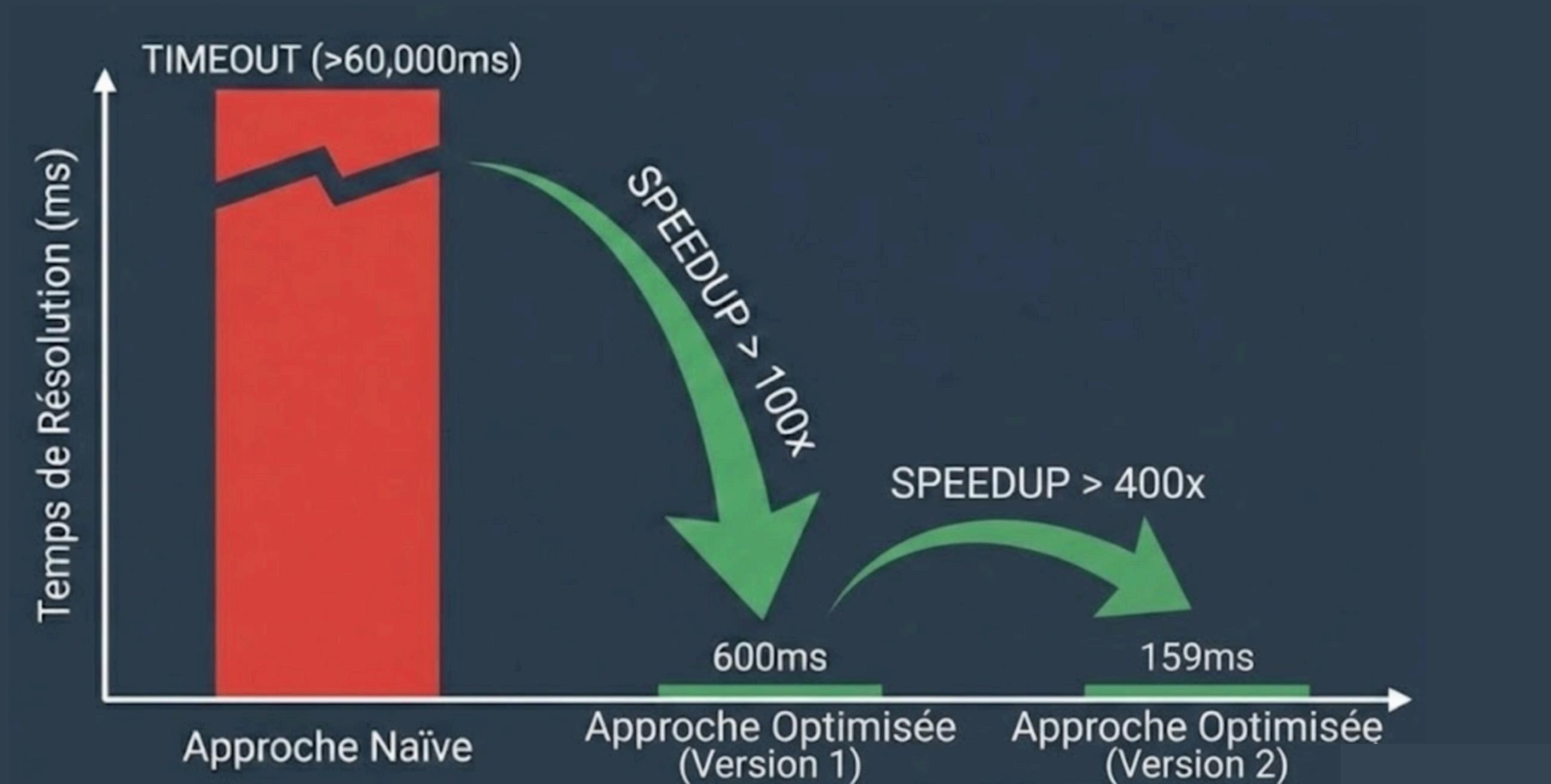
A 9x9 Sudoku grid with several numbers filled in. The word "TIMEOUT" is written diagonally across the grid in large red letters.

	6	3	1					
3		2	5	1	6	7	5	1
	7	8	4	5			1	6
6				4	3	6		
4	7	1	5	8		9	5	2
	6	1	2		7	8	3	
4			3	5	8		9	
9	2	3	6			4		
		7			8	4		
3		3		5		5	4	1
	2	7		5		6	2	7
	1		8	2	7	7		5
4	8		5	6	7	1	2	3
	6	3	2	5	4	2	7	1
2		7	4		3		7	
1	3							
			1					

# HARD 25x25

## 25<sup>625</sup> états

# LE RÉSULTATS



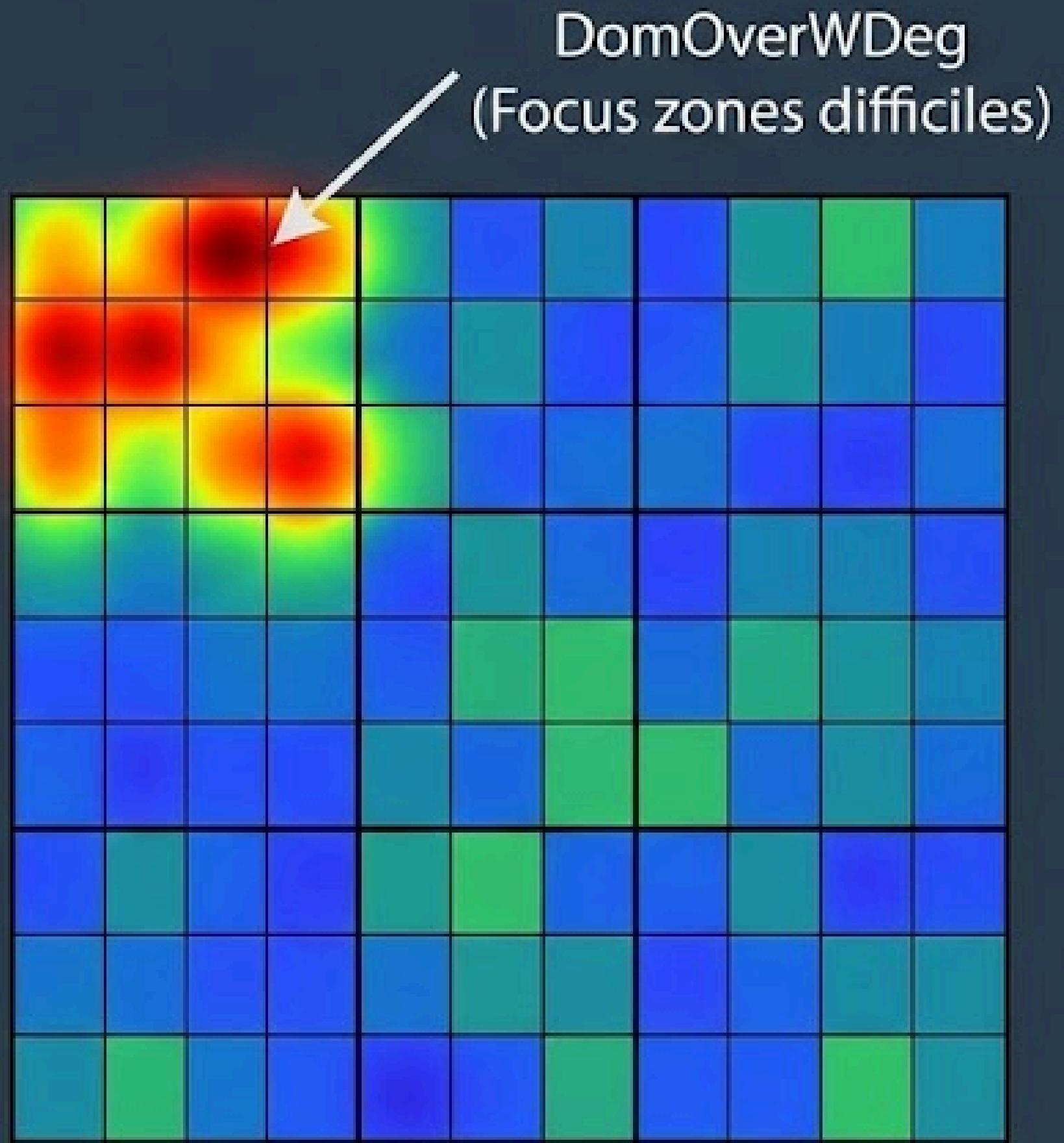
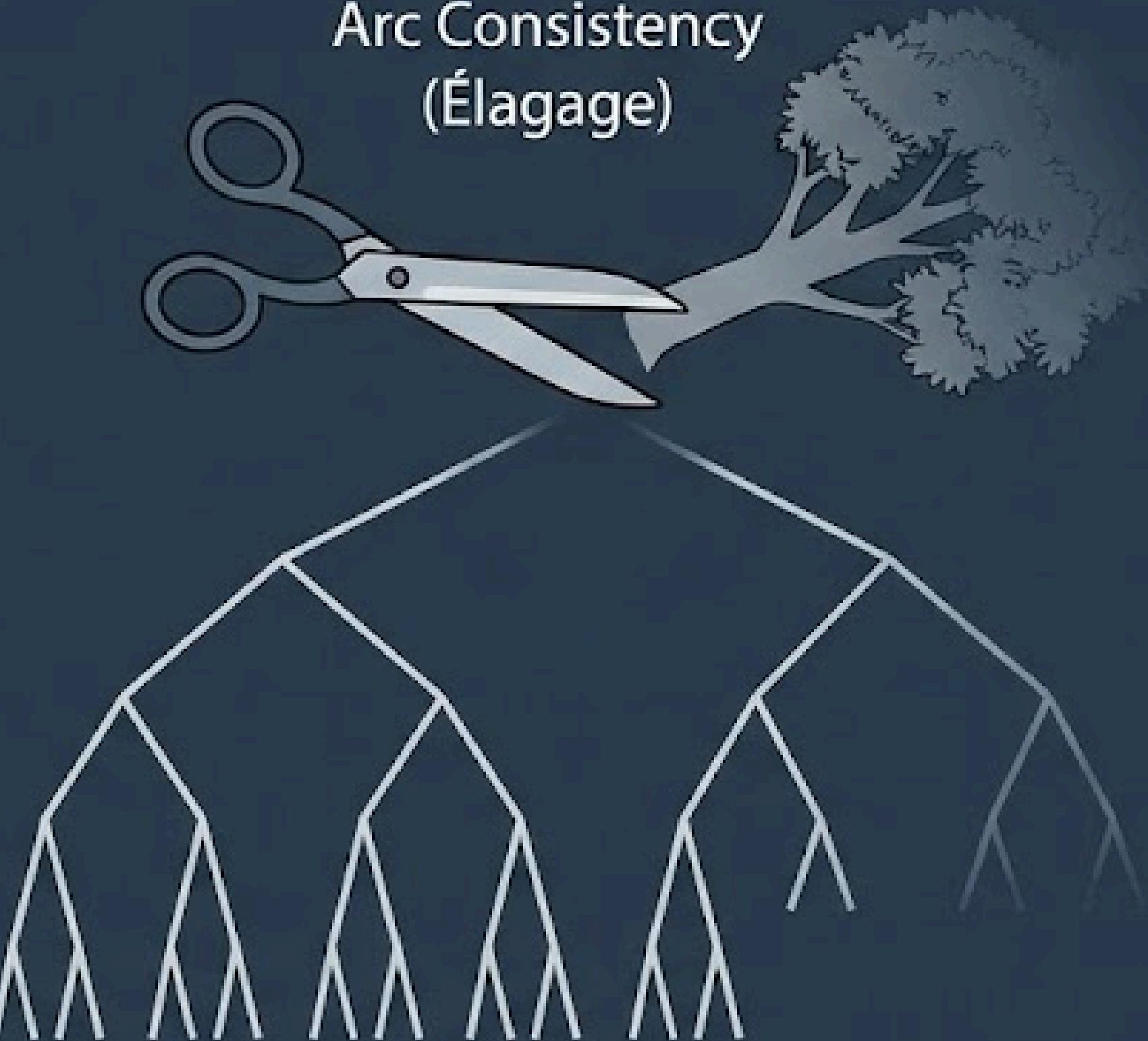
INPUTORDER + DEFAULT  
CONSISTENCY + VALEUR MIN (DEFAULT) + NO RESTART

DOMOVERWDEG + ARC  
CONSISTENCY (AC) + VALEUR MIN (PAR DÉFAUT) + LUBY RESTARTS

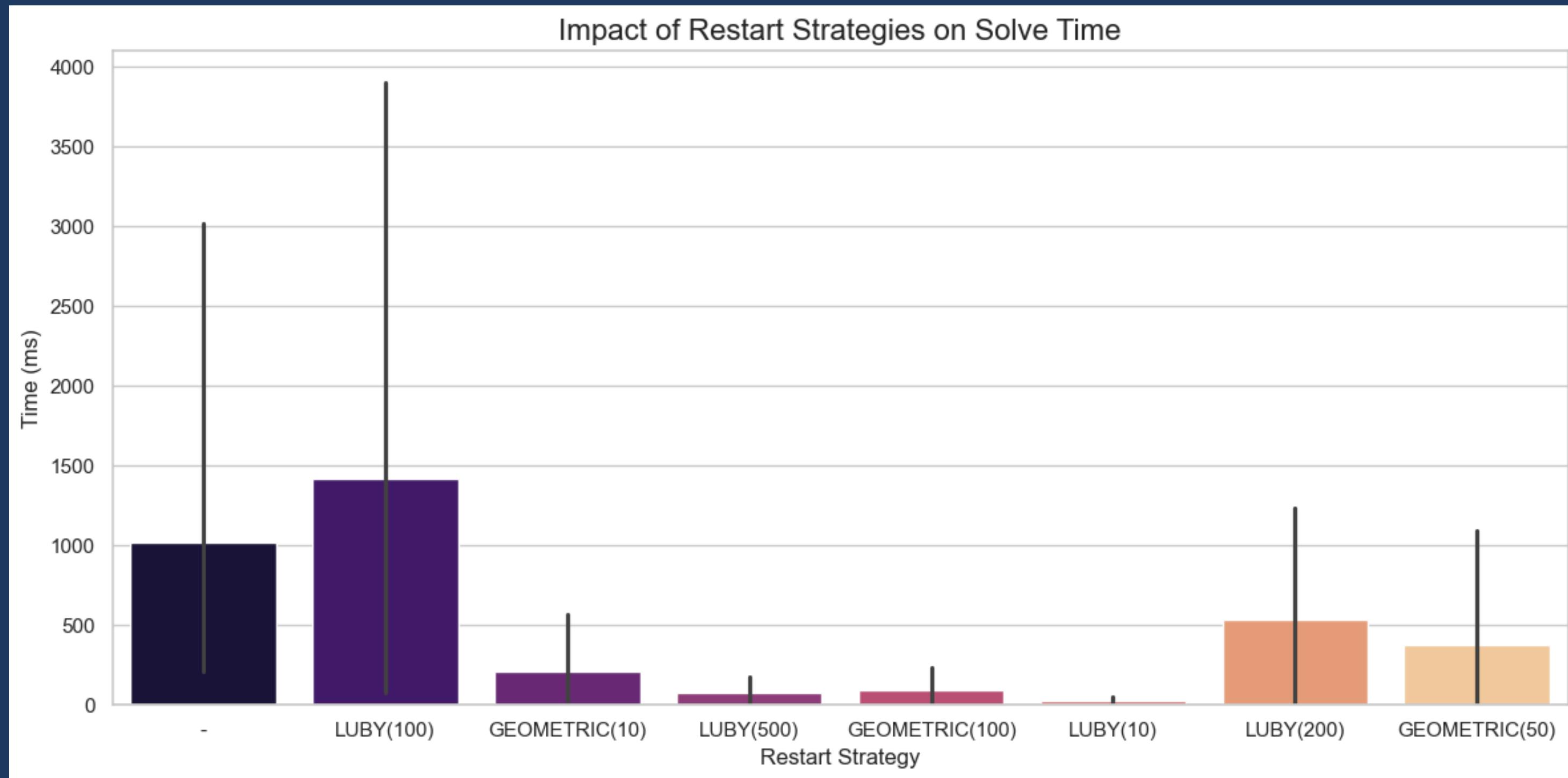
DOMOVERWDEG + ARC  
CONSISTENCY (AC) + VALEUR RANDOM + LUBY RESTARTS (BASE 10).

# ANALYSE DÉTAILLÉE

Arc Consistency  
(Élagage)



# Impact de la Stratégie de Redémarrage



# **III - L'approche incomplète**

# Backtracking Intelligent & Heuristiques

## Heuristique MRV

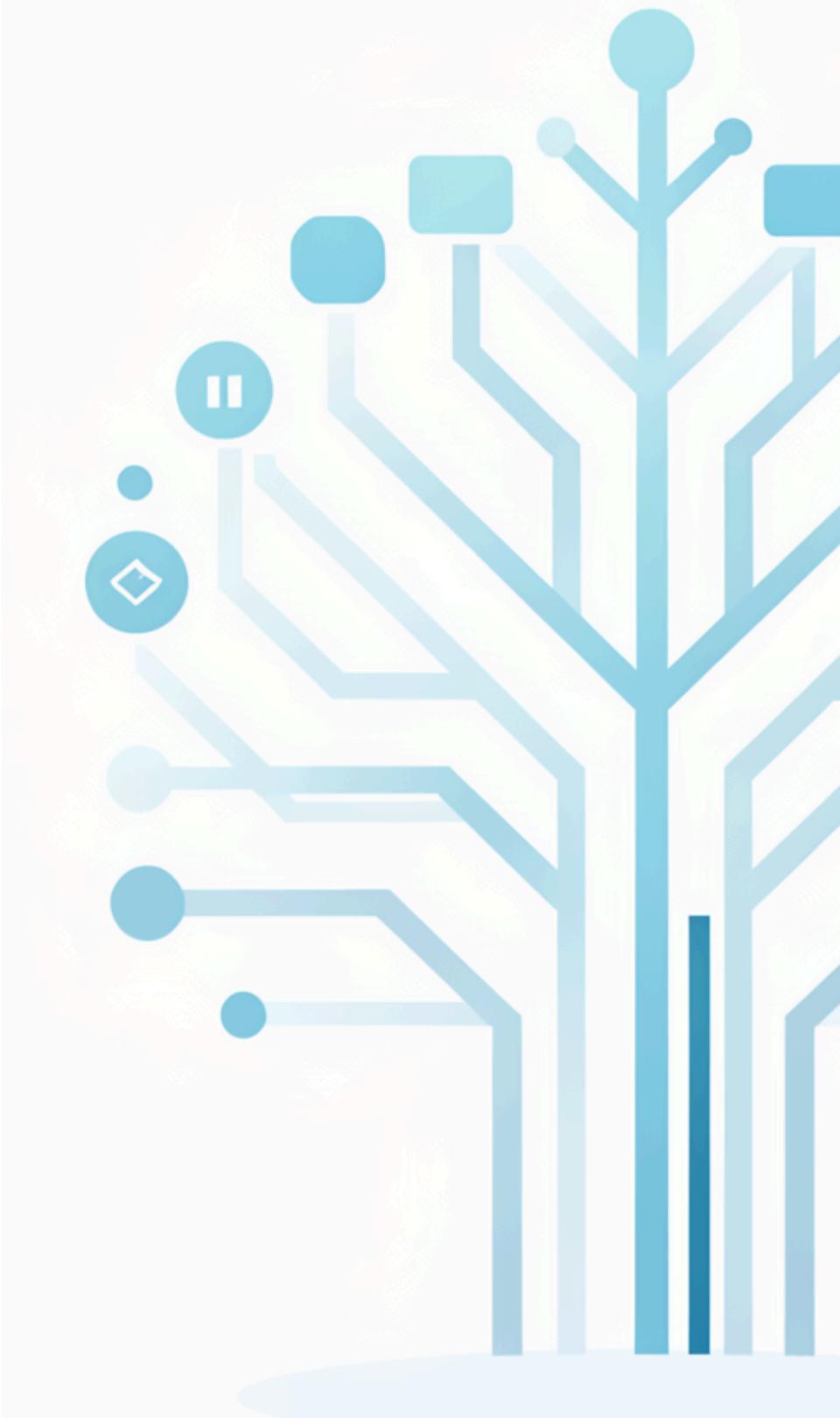
L'algorithme identifie la case la plus critique avec le moins de choix possibles, réduisant drastiquement l'arbre de recherche.

## Choix Stratégique

Au lieu de remplir de gauche à droite, on traite les cas contraints en premier pour éviter des milliers de mauvaises directions.

## Performance

Sécurité intégrée avec maxIterations à 200 000. Pour 99% des grilles classiques, solution trouvée en quelques millisecondes.



# Propagation & 'Naked Singles'



## Placement Initial

Quand un chiffre est placé, il est rayé instantanément des possibilités des cases voisines (ligne, colonne, carré).

## Détection Automatique

Si une case voisine a une seule possibilité restante (Naked Single), elle est validée immédiatement sans deviner.

## Réaction en Chaîne

Cette validation déclenche d'autres nettoyages. Sur grilles faciles, la propagation résout parfois tout sans backtracking.

# Custom Undo Stack

## Le Problème

Les implémentations classiques copient la grille entière à chaque essai, saturant la mémoire.

## Notre Solution

Une Undo Stack personnalisée avec classe interne Change qui empile uniquement l'information minimale ( coordonnées + ancienne valeur ).



Lors du backtrack, on dépile simplement ces petits objets pour restaurer l'état précédent. Résultat : pas de saturation mémoire.

# Résultats

