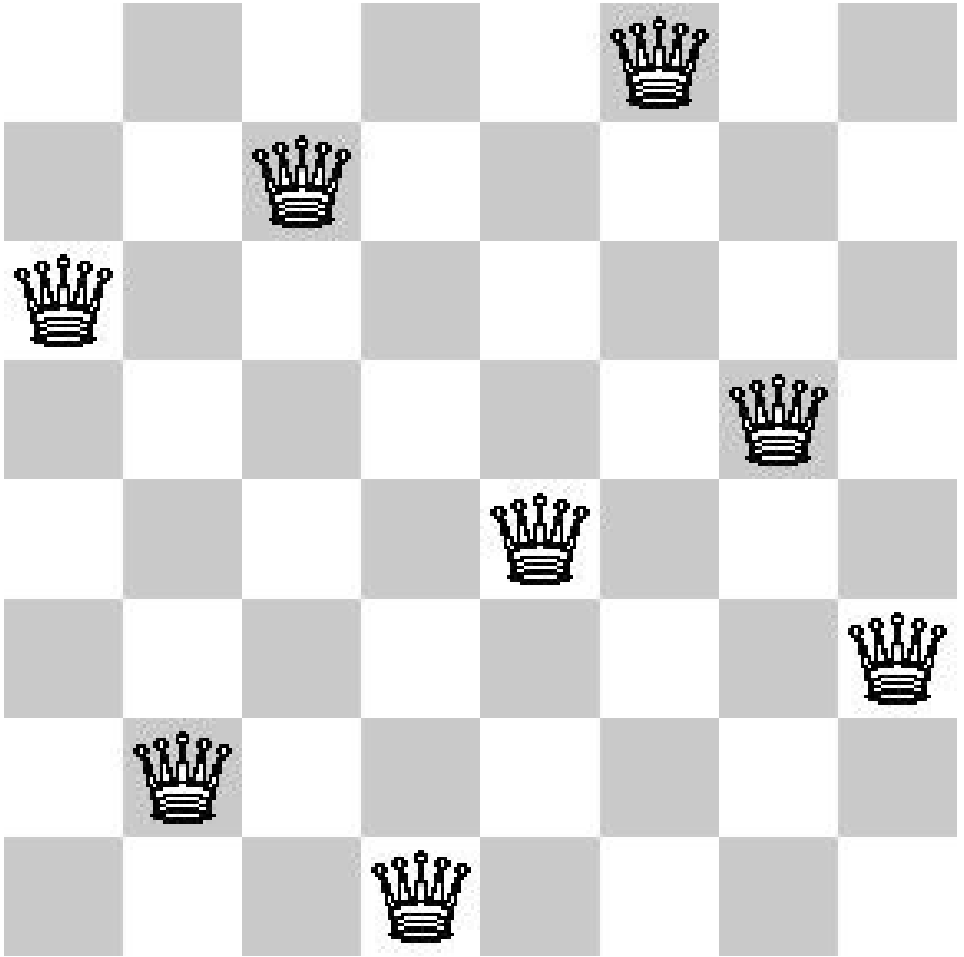
# Résolution de problème

# Introduction



# Représentation CSP

N reines à disposer sur un plateau de taille NxN

Variables :
Xi avec i ∈ {1, 2, …, n}, reine sur la ligne i
Xi a pour domaine {1, 2, …, n}, la colonne où elle est placée.

Contraintes :
$Xi \neq Xj$

$i \neq j$

$| Xi - i | \neq | Xj - j |$

# Représentation Objet

## Plateau

Représenté sous forme de tableau de tableau d'entier

## Domaine

Représenté sous forme d'une List<List<Integer>>

Chaque reine possède une liste de position possible

# Représentation Objet - Améliorations

plateau[i] = y correspond à la reine de la ligne i dans la colonne y

Pas de vérification par ligne

**Gestion des erreurs trop lourdes**
(80 % du temps de traitement)

-Réduction du nombre d'appel au calcul des erreurs

-Amélioration de la contrainte diagonale :

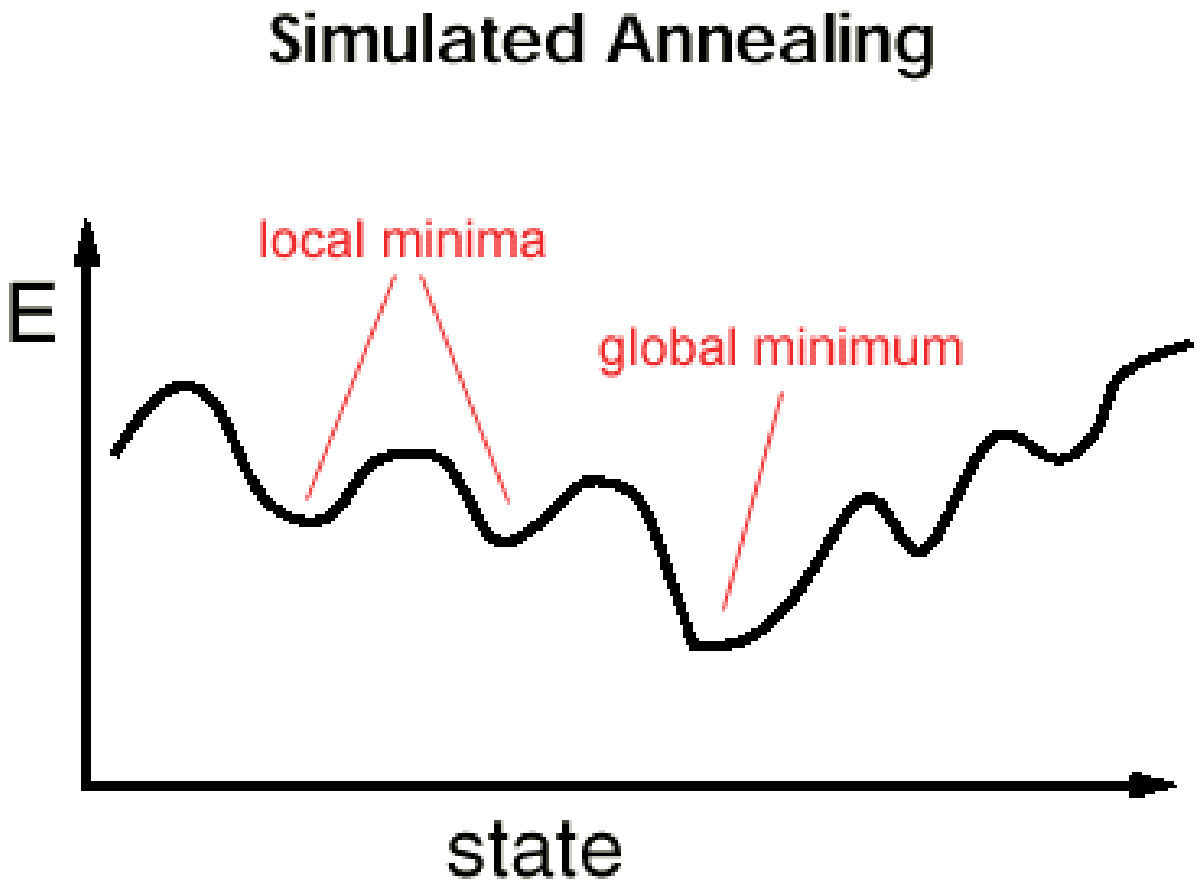
$Xi - Xj \neq i - j$ et $Xj - Xi \neq j - i$

$|Xi - i| \neq |Xj - j|$

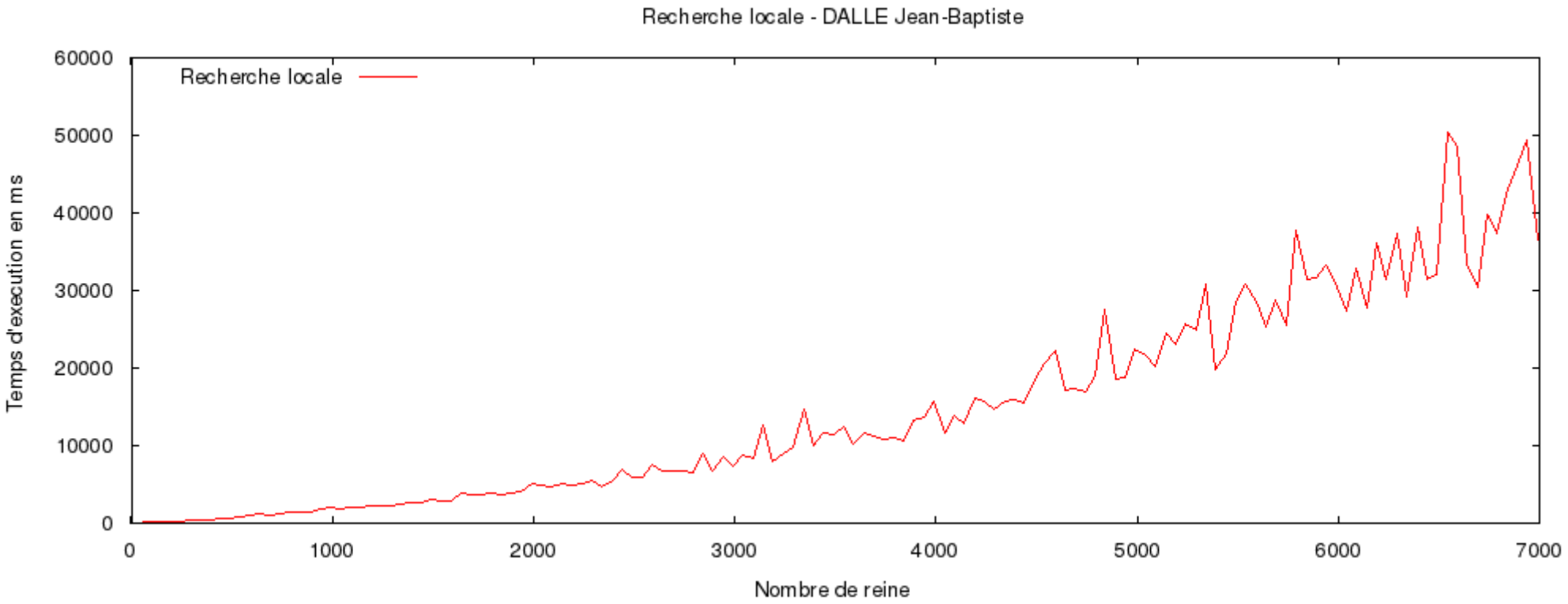
-Représentation des erreurs sous forme d'une liste de conflit entre deux reines

-Représentation des erreurs sous forme d'un compteur

# Recherche locale – recuit simulé



# Recherche locale



# Programmation par contraintes

Méthode de résolution implémentées :
-Back Tracking
-Forward Checking
-Back Tracking + Arc Consistancy
-Forward Checking + Arc Consistancy

Heuristique implémentées :
-Premier trouvé
-Minimum
-Maximum
-Random
-Domaine minimum

# Back Tracking

On place une reine sur une case sans danger

On continue à placer des dames tant que cela est possible

On retourne en arrière si aucune solution n'existe

# Forward Checking

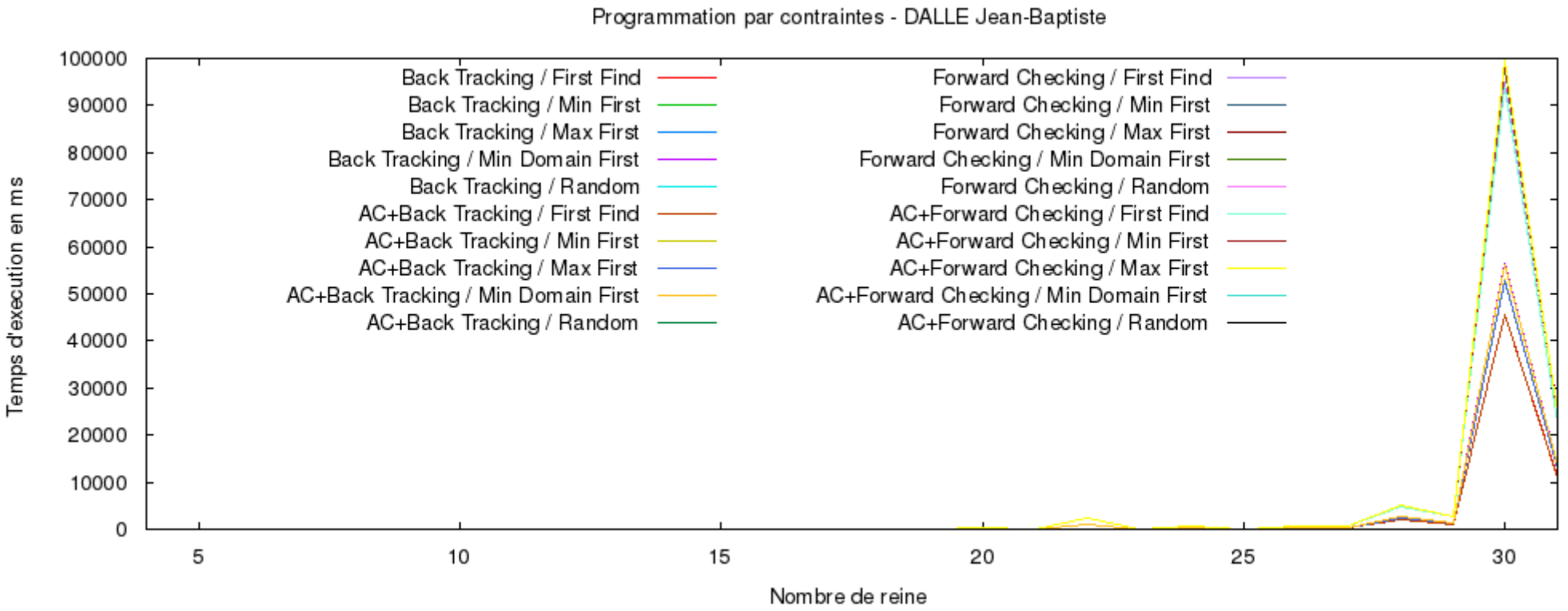
On place une reine sur une case sans danger

On réduit le domaine des autres dames en conséquences

On continue à placer des dames tant que cela est possible

On retourne en arrière si aucune solution n'existe tout en restaurant le domaine

# Programmation par contraintes



# Random et Domaine minimum