Présentation 5
PSAR

David TOTY

Maxime TRAN

Objectif:
Modéliser et résoudre le problème du Sudoku

Rappel: Programmation par contraintes

En programmation par contraintes, un problème doit être formulé à l'aide des notions suivantes :

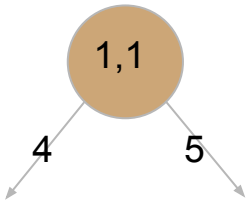
- Des ***variables***, des ***domaines*** et des ***contraintes***.

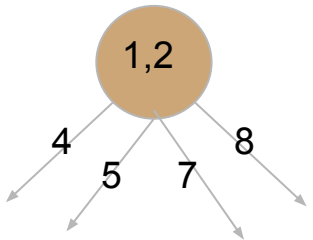
Exemple:

Variables: Les 81 cases du Sudoku
Domaines: L'ensemble {1, 2, 3, 4, 5, 6, 7, 8, 9}
Contraintes: 1 seule occurrence de valeur sur chaque ligne, colonne et région

Comment résoudre un sudoku à la main

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





Stratégie de recherche: Recherche Adaptative

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

X = {4}

Algorithme adaptatif

Tableau du Sudoku:

0 6 0 0 5 0 0 2 0
0 0 0 3 0 0 0 9 0
7 0 0 6 0 0 0 1 0
0 0 6 0 3 0 4 0 0
0 0 4 0 7 0 1 0 0
0 0 5 0 9 0 8 0 0
0 4 0 0 0 1 0 0 6
0 3 0 0 0 8 0 0 0
0 2 0 0 4 0 0 5 0

Tableau de poids:

5 3 4 5 2 3 2 4 4
5 3 3 5 3 3 3 4 4
6 3 4 4 2 3 2 3 4
4 4 5 4 3 2 4 1 4
4 2 4 3 3 3 5 2 4
3 2 4 3 3 3 4 3 3
3 4 3 4 1 5 4 3 5
4 4 3 4 2 5 3 2 5
4 4 4 2 1 4 3 3 5

Tableau de parcours: (i, x, y, poids)

0 : 3 : 7 : 1
1 : 6 : 4 : 1
2 : 8 : 4 : 1
3 : 0 : 4 : 2
4 : 0 : 6 : 2
5 : 2 : 4 : 2
6 : 2 : 6 : 2
7 : 3 : 5 : 2
8 : 4 : 1 : 2
9 : 4 : 7 : 2
...

Algorithme adaptatif

```
public void initPoids();

public void initParcours();

public void initFait();

public void solve(int row, int col)

public void solve2(int row, int col,int xx);
```

Tableaux:

- Possibilité
- Etages

Algorithme adaptatif

public void initPoids();

public void initParcours();

public void initFait();

public void solve(int row, int col)

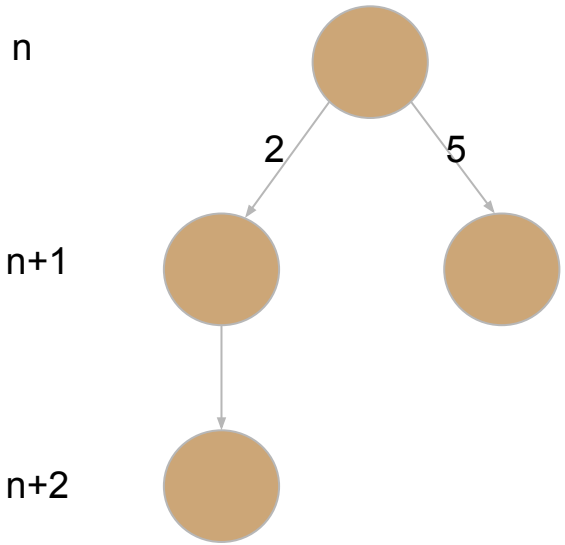
public void solve2(int row, int col,int xx);

Tableaux:

- Possibilité
- Etages

Tableau etages stock n.

Tableau de possibilités stock la valeur 5.



Algorithme adaptatif

```
public void solve2(int row, int col,int xx) {
        int cpt=0;
        int premierPoss=0;
        int dernierPoss=0;
        int profondeur=0;
        if (etage>79){
                System.out.println("Fin ");
                System.out.println(this);
                Return;
        }
        if (model[row][col] != 0){
                etage++;
                solve3(parcours[0][etage],parcours[1][etage],1);
        }
        ...
```

Algorithme adaptatif

```
else {
        for (int num = xx; num < TAILLE+1; num++) {
                if (checkRow(row, num) && checkCol(col, num) && checkBox(row, col, num)) {
                        cpt++;
                        if(cpt==1)
                                premierPoss=num;
                        if (cpt>1){
                                possibilites.add(num);
                                numpossibilites.add(etage);
                                break;
                        }
                }
        }
```

Algorithme adaptatif

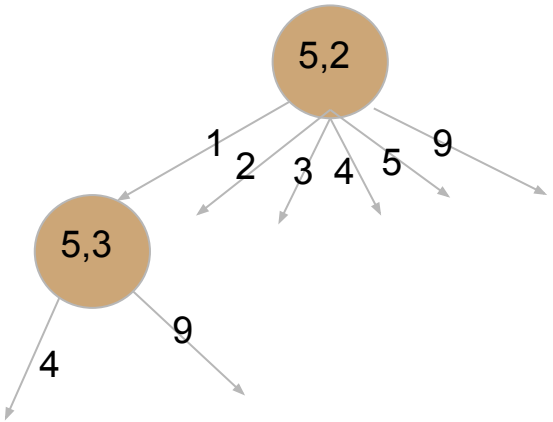
```
if (cpt > 0){ //On a une possibilite, on l'essaye
        model[row][col] = premierPoss;
        Etage++;
        solve3(parcours[0][etage],parcours[1][etage],1);
}
else { // Pas de possibilite (incoherence)
        profondeur = etage - (numpossibilites.get(numpossibilites.size()-1));
        for(int i = 0; i<= profondeur; i++){
                if (fait[(parcours[0][etage-i])][(parcours[1][etage-i])] == false){
                model[parcours[0][etage-i]][parcours[1][etage-i]]=0;
                }
        }
        ...
```

Algorithme adaptatif

```
            dernierPoss = possibilites.get(possibilites.size()-1);
            Etage = numpossibilites.get(numpossibilites.size()-1);
            numpossibilites.remove(numpossibilites.size()-1);
            possibilites.remove(possibilites.size()-1);
             solve3(parcours[0][etage],parcours[1][etage],dernierPoss);
        }
}
```

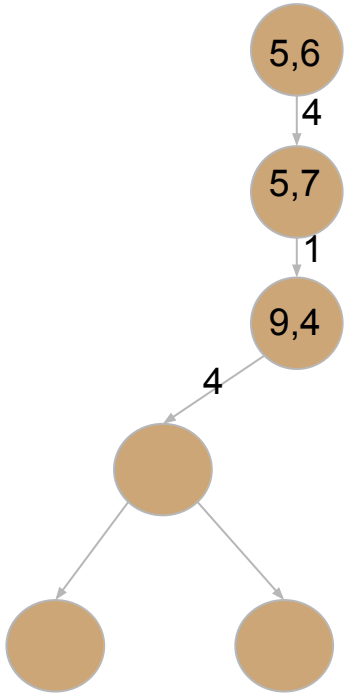
Comparaison de parcours de solution

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



Comparaison de parcours de solution

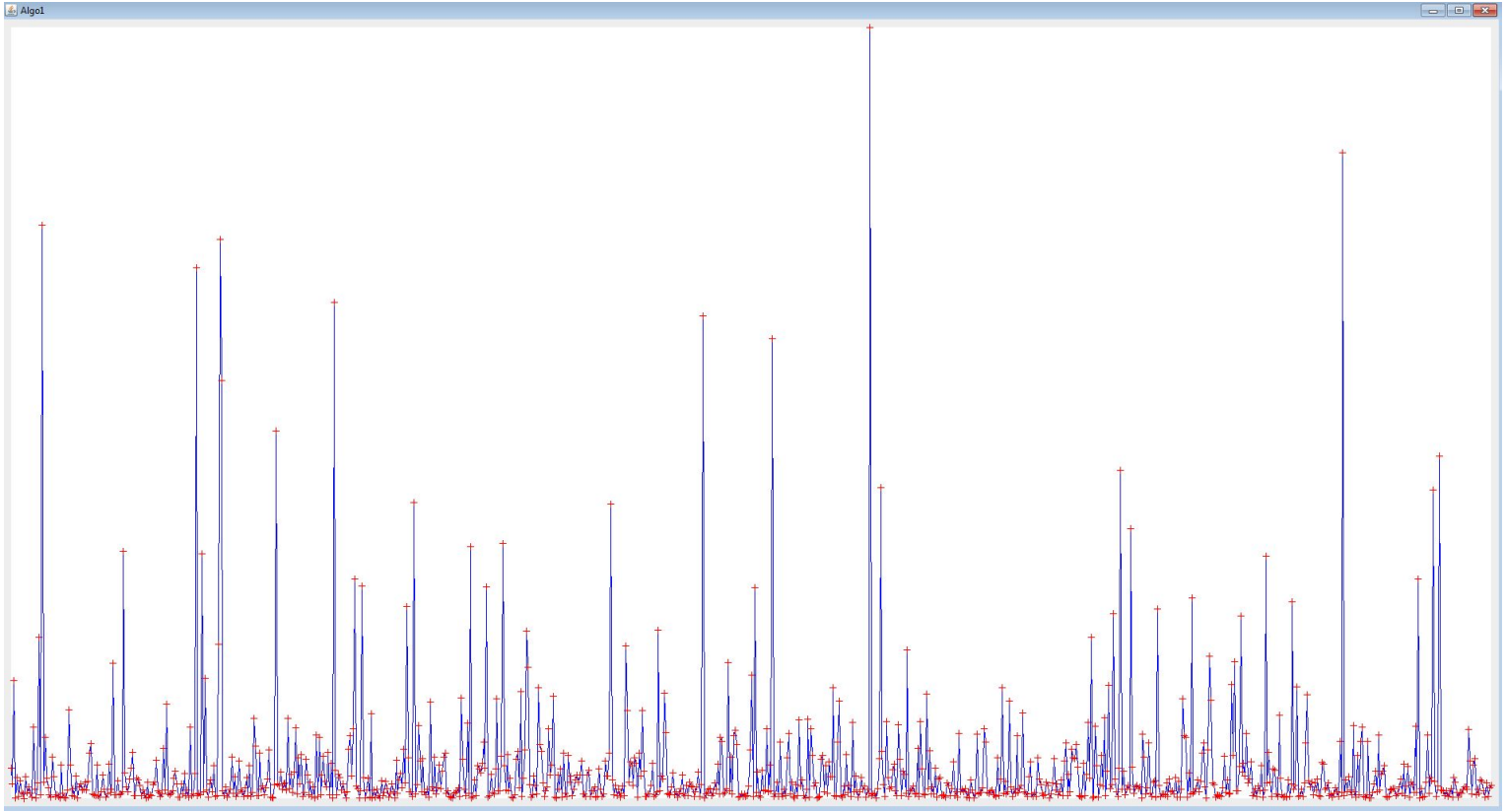
		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7					x	x		8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5	x	1		3		



Jeu de test

```
 *** Resolutions de 1000 Sudokus

Temps total algo 1 : 8472164443 nanosecondes
Temps moyen algo 1 : 8472164 nanosecondes
Temps max algo 1 : 214837852 nanosecondes
Temps min algo 1 : 29719 nanosecondes

*   **********   *

Temps total algo 2 : 43094226 nanosecondes
Temps moyen algo 2 : 43094 nanosecondes
Temps max algo 2 : 4314949 nanosecondes
Temps min algo 2 : 13809 nanosecondes
```

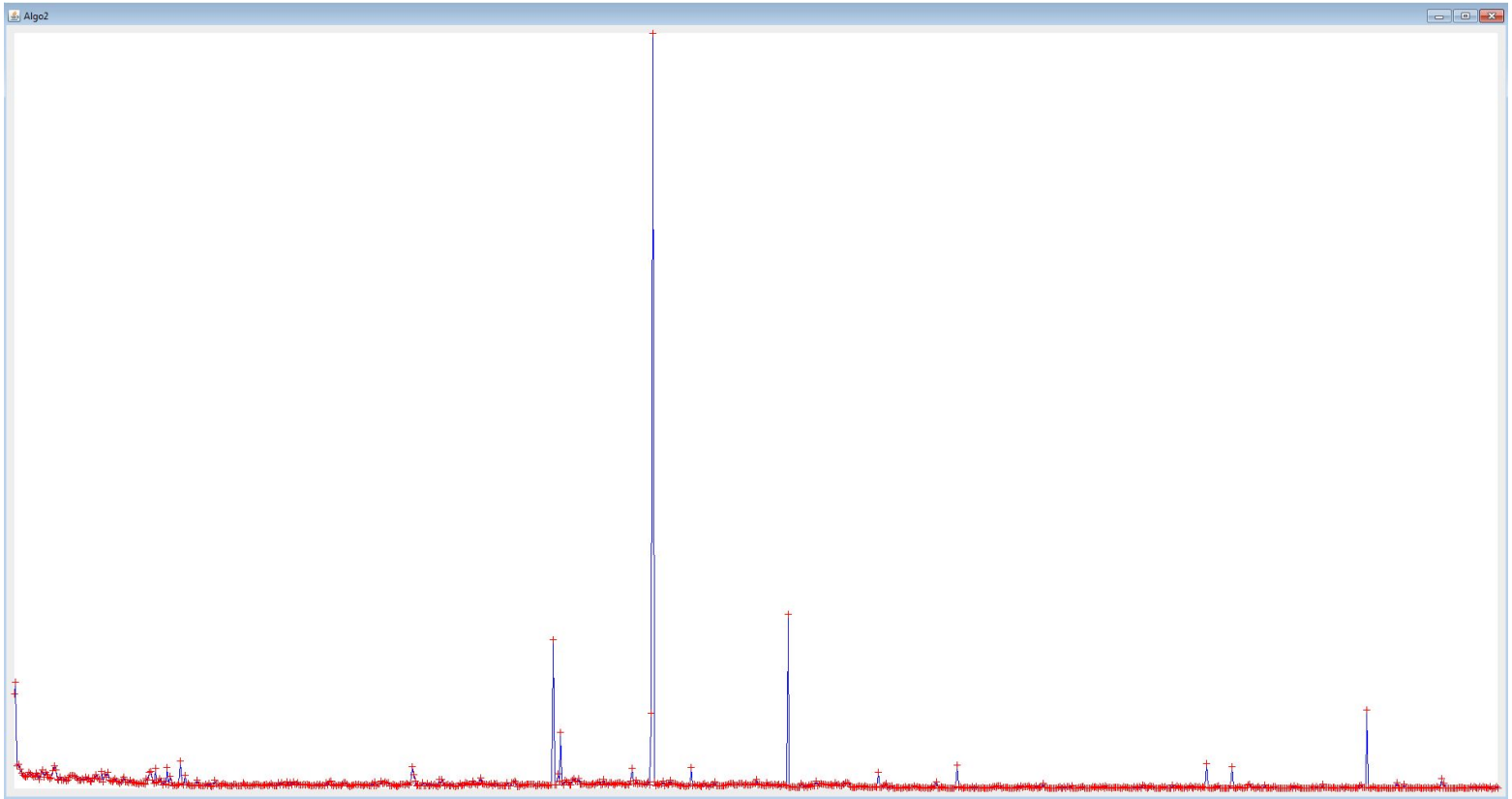
Graphe - Algorithme classique



Abscisse:
Numéro de Sudoku

Ordonnée:
Temps en ns

Graphe - Algorithme adaptative



Abscisse:
Numéro de Sudoku

Ordonnée:
Temps en ns

Modèle du problème de Sudoku sous MiniZinc

```
include “alldiferent.mzn”;

set of int: domaine = 1..9;
array [domaine, domaine] of int: grille;
grille =
[ | 0, 0, 0, 0, 0, 0, 0, 0, 0
  | 0, 6, 8, 4, 0, 1, 0, 7, 0
  | 0, 0, 0, 0, 8, 5, 0, 3, 0
  | 0, 2, 6, 8, 0, 9, 0, 4, 0
  | 0, 0, 7, 0, 0, 0, 9, 0, 0
  | 0, 5, 0, 1, 0, 6, 3, 2, 0
  | 0, 4, 0, 6, 1, 0, 0, 0, 0
  | 0, 3, 0, 2, 0, 7, 6, 9, 0
  | 0, 0, 0, 0, 0, 0, 0, 0, 0
|];
```

```
array [domaine, domaine] of var domaine: puzzle;

constraint forall (i, j in domaine) (
        If grille [i, j] > 0 then puzzle [i, j] = grille [i, j] else true endif );


predicate ligne_diff (int: ligne) =
        alldifferent (colonne in domaine) (puzzle[ligne, colonne]);


predicate colonne_diff (int: colonne) =
        alldifferent (ligne in domaine) (puzzle[ligne, colonne]);
```

Modèle du problème de Sudoku sous MiniZinc

```
predicate region_diff (int: ligne, int: colonne) =
        alldifferent (i, j in 0..2) (puzzle[ligne + i, colonne + j]);

constraint forall (ligne in domaine)              (ligne_diff (ligne));
constraint forall (colonne in domaine)        (colonne_diff (colonne));
constraint forall (ligne, colonne in {1, 4, 7}) (region_diff (ligne, colonne));

solve satisfy;

output [ if j = 1 then "\n" else " " endif ++
        Show (puzzle[i,j]) | i,j in domaine] ++ ["\n"];
```

Explication du code

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

```
predicate region_diff (int: ligne, int: colonne) =
        alldifferent (i, j in 0..2) (grille[ligne + i, colonne + j]);

constraint forall (ligne, colonne in {1, 4, 7})
        (region_diff (ligne, colonne))
```

Explication du code

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

```
predicate region_diff (int: ligne, int: colonne) =
        alldifferent (i, j in 0..2) (grille[ligne + i, colonne + j]);

constraint forall (ligne, colonne in {1, 4, 7})
        (region_diff (ligne, colonne));
```

Exemple:

region(1,4)

Explication du code

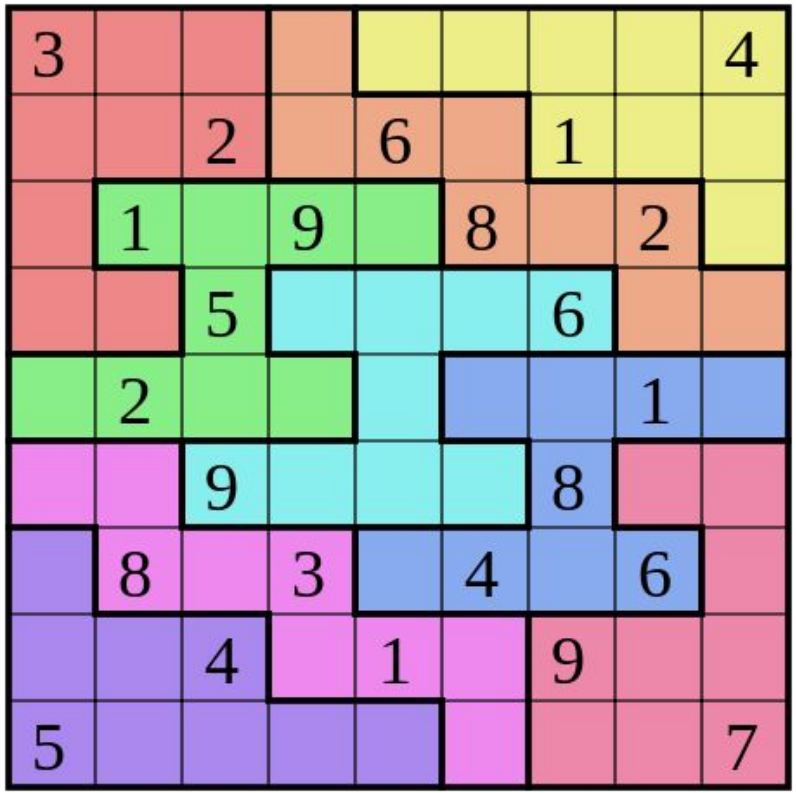
		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
7								8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1		3		

```
predicate region_diff (int: ligne, int: colonne) =
        alldifferent (i, j in 0..2) (grille[ligne + i, colonne + j]);

constraint forall (ligne, colonne in {1, 4, 7})
        (region_diff (ligne, colonne));
```

Exemple:

region(1,4)

Chaos Sudoku



Modèle du problème de Chaos Sudoku sous MiniZinc

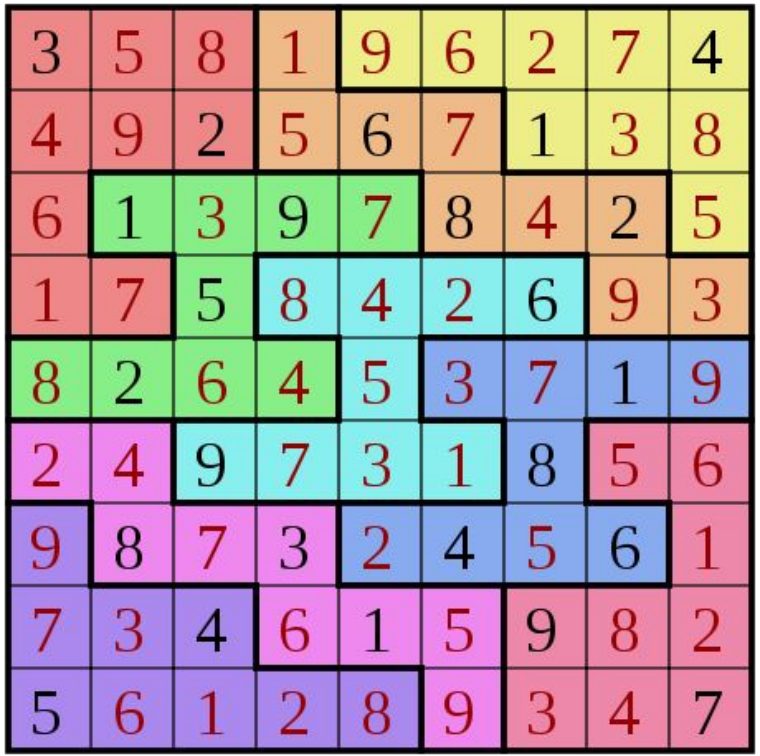
```
array [domaine, domaine] of int: grille;

grille =
[| 3, 0, 0, 0, 0, 0, 0, 0, 4
| 0, 0, 2, 0, 6, 0, 1, 0, 0
| 0, 1, 0, 9, 0, 8, 0, 2, 0
| 0, 0, 5, 0, 0, 0, 6, 0, 0
| 0, 2, 0, 0, 0, 0, 0, 1, 0
| 0, 0, 9, 0, 0, 0, 8, 0, 0
| 0, 8, 0, 3, 0, 4, 0, 6, 0
| 0, 0, 4, 0, 1, 0, 9, 0, 0
| 5, 0, 0, 0, 0, 0, 0, 0, 7
|];
```

```
array [domaine, domaine] of int: region;

region =
[| 1, 1, 1, 2, 3, 3, 3, 3, 3
| 1, 1, 1, 2, 2, 2, 3, 3, 3
| 1, 4, 4, 4, 4, 2, 2, 2, 3
| 1, 1, 4, 5, 5, 5, 5, 2, 2
| 4, 4, 4, 4, 5, 6, 6, 6, 6
| 7, 7, 5, 5, 5, 5, 6, 9, 9
| 8, 7, 7, 7, 6, 6, 6, 6, 9
| 8, 8, 8, 7, 7, 7, 9, 9, 9
| 8, 8, 8, 8, 8, 7, 9, 9, 9
|];
```

Modèle du problème de Chaos Sudoku sous MiniZinc

```
predicate region_chaos (int: r) =
        (alldifferent ([  puzzle[ligne,colonne] | ligne,colonne in domaine where region[ligne,colonne]==r ]));

constraint forall (r in domaine)
        (region_chaos (r));
```

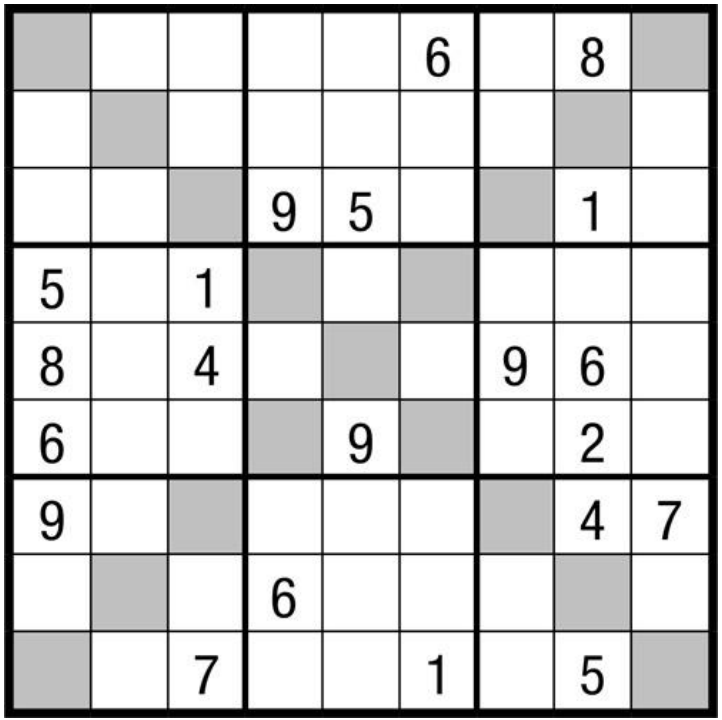
Résultat



> mzn-g12fd chaos_sudoku.mzn

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

Ajouter des contraintes sur les diagonales



constraint (alldifferent ([puzzle[ligne, ligne] | ligne in domaine]));

constraint (alldifferent ([puzzle[10-ligne, ligne] | ligne in domaine]));

Questions ?

Merci !
Pour nous contacter:

david.toty@etu.upmc.fr

maxime.tran@etu.upmc.fr

Encadrant du projet:

Responsable: Fabrice KORDON
Client: Tarek Menouer