Présentation 4
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 :

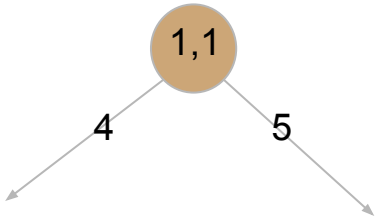
- Des ***variables***, des ***domaines*** et des ***contraintes***.

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

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		



Modélisation de la recherche adaptative

```
public SimplifiedSudoku() {
    model = new int[][] {{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}};
    first=true;

    poidsTotal = new int[][] {{0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0},
            {0, 0, 0, 0, 0, 0, 0, 0, 0}};


}
```

Modélisation de la recherche adaptative

```
/** Checks if num is an acceptable value for the given row */
protected boolean checkRow(int row, int num) {
        for (int col = 0; col < 9; col++)
                if (model[row][col] == num)
                        return false;

        return true;
}

/** Checks if num is an acceptable value for the given column */
protected boolean checkCol(int col, int num) {
        for (int row = 0; row < 9; row++)
                if (model[row][col] == num)
                        return false;

        return true;
}

/** Checks if num is an acceptable value for the box around row and col */
protected boolean checkBox(int row, int col, int num) {
        row = (row / 3) * 3;
        col = (col / 3) * 3;

        for (int r = 0; r < 3; r++)
                for (int c = 0; c < 3; c++)
                        if (model[row + r][col + c] == num)
                                return false;

        return true;
}
```

Modélisation de la recherche adaptative

```java
/** Recursive function to find a valid number for one single cell */
public void solve(int row, int col){
        /* Dernière ligne on s'arrète */
        if (row> 8){
                System.out.println(this);
                return;
        }

        // If the cell is not empty, continue with the next cell
        if (model[row][col] != 0)
                next(row, col);
        else {
                // Find a valid number for the empty cell
                for (int num = 1; num < 10; num++) {
                        if (checkRow(row, num) && checkCol(col, num) && checkBox(row, col, num)) {
                                model[row][col] = num;

                                // Delegate work on the next cell to a recursive call
                                next(row, col);
                        }
                }
                // No valid number was found, clean up and return to caller
                model[row][col] = 0;
        }
}

public void next(int row, int col) {
        if (col < 8)
                solve(row, col + 1);
        else
                solve(row + 1, 0);
}
```

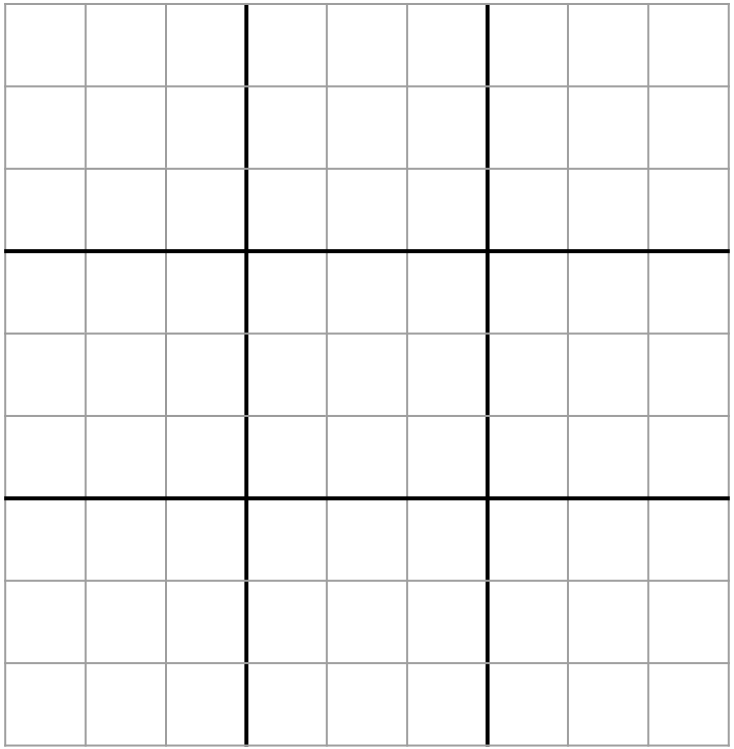
Modélisation de la recherche adaptative

```
public void initPoids() {
    int poids=0;

    for (int i=0;i<9;i++){
        for (int j=0;j<9;j++){
            for(int k=0;k<9;k++){
                if(checkRow(i,k) && checkCol(j,k) && checkBox(i,j,k)){
                    poids++;
                }
            }
            poidsTotal[i][j]=poids;
            poids=0;
        }
    }
    System.out.println("******************");
    for(int i=0;i<9;i++){
        for(int j=0;j<9;j++)
            System.out.print(poidsTotal[i][j] +" ");
        System.out.println();
    }
    System.out.println("******************");


}

public void solve2(int row, int col){
    initPoids();

    if (model[row][col] != 0)
        next2(row, col);


        for (int num = 1; num < 10; num++) {
            if (checkRow(row, num) && checkCol(col, num) && checkBox(row, col, num)) {
                model[row][col] = num;
                next2(row, col);
            }
        }
        //model[row][col] = 0;

}
```

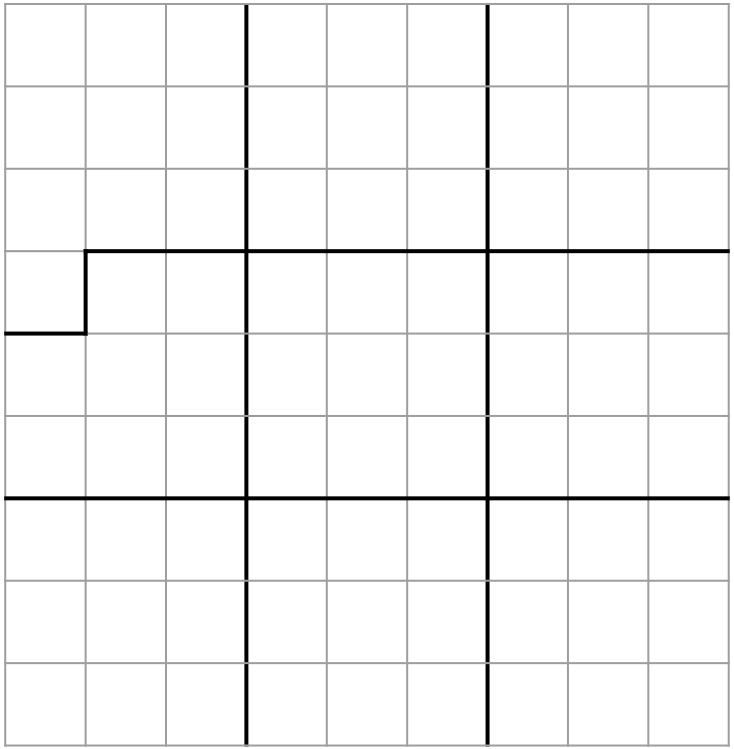
Modélisation de la recherche adaptative

```
public void next2(int row, int col) {
    int min=9;
    int imin=0;
    int jmin=0;

    for (int i=0;i<9;i++){
        for(int j=0;j<9;j++){
            if(poidsTotal[i][j]==1 && model[i][j] == 0){
                System.out.println("je resoud la case " +i +" "+ j);
                System.out.println(this);
                solve2(i,j);
                //return;
            }
            if(poidsTotal[i][j]<min && model[i][j] == 0){
                min=poidsTotal[i][j];
                imin=i;
                jmin=j;
            }
        }
    }
    System.out.println("je resoud la case " +imin +" "+ jmin);
    System.out.println(this);
    solve2(imin, jmin);
}
```

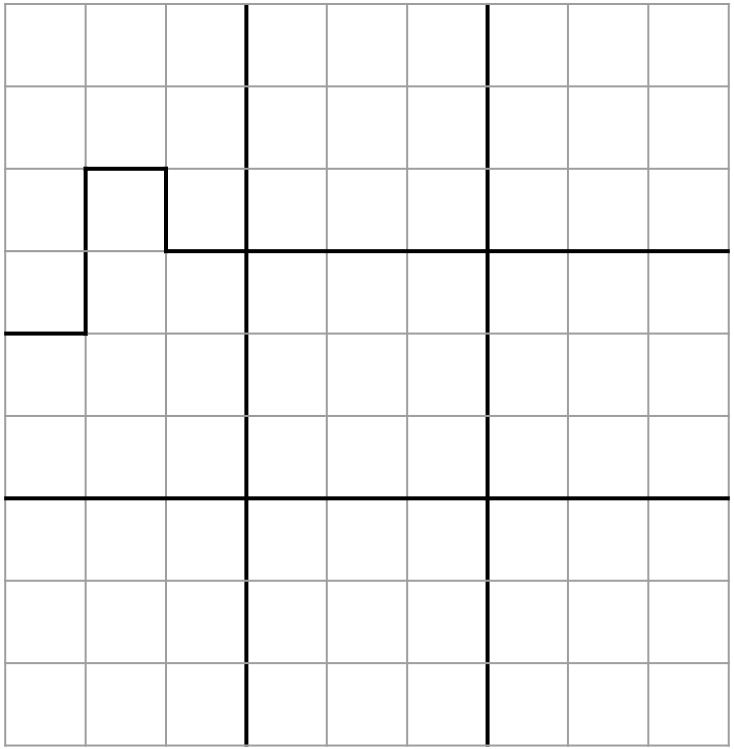
Interprétation pour la contrainte sur les régions



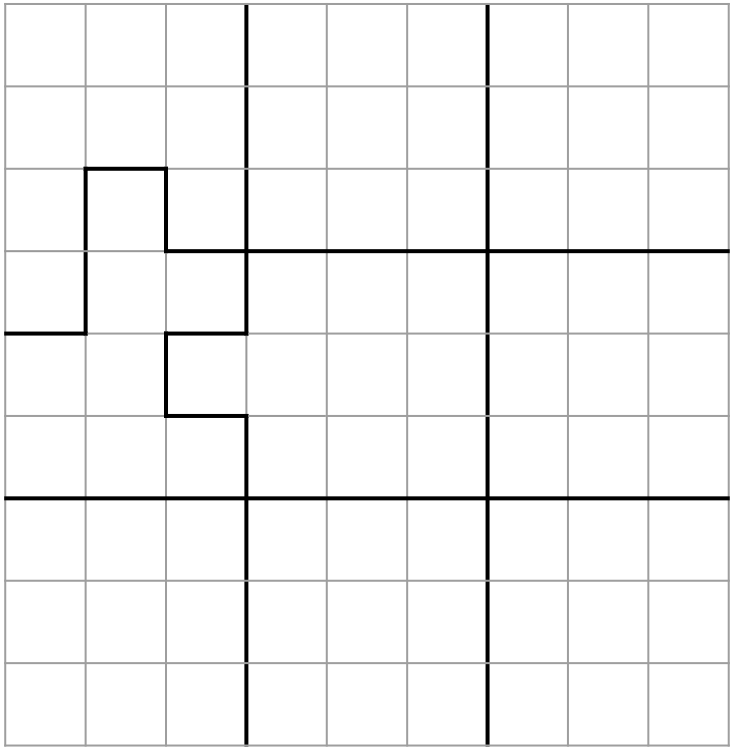
Interprétation pour la contrainte sur les régions



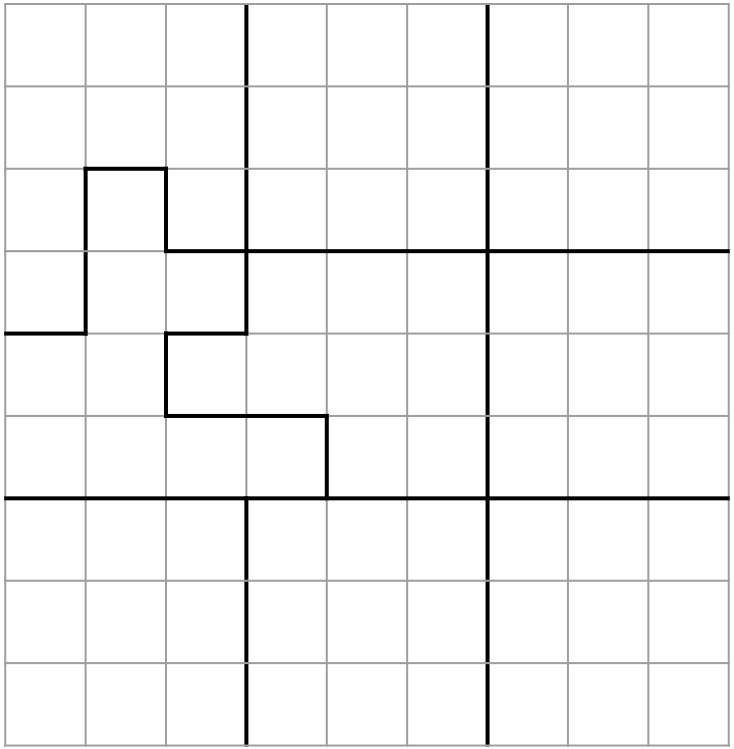
Interprétation pour la contrainte sur les régions



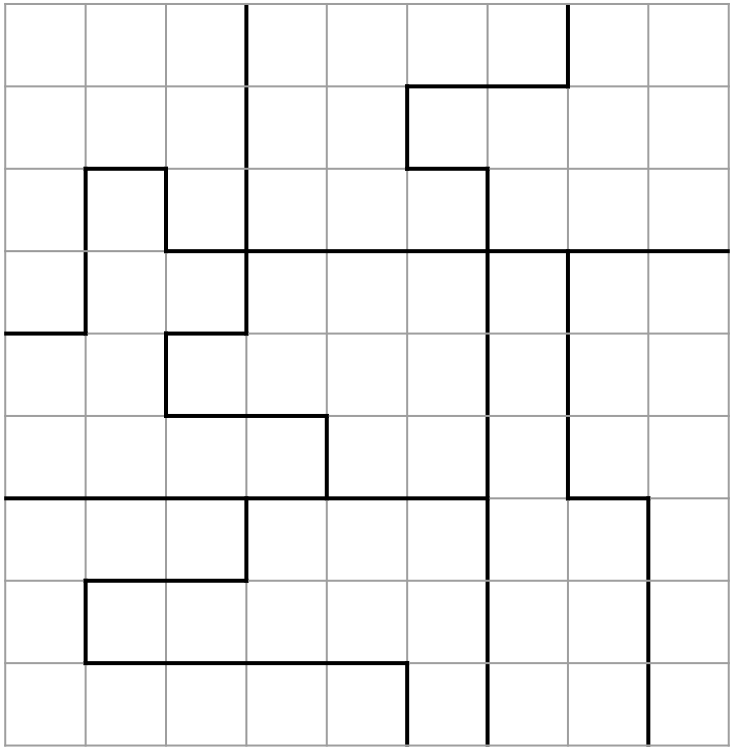
Interprétation pour la contrainte sur les régions



Interprétation pour la contrainte sur les régions



Interprétation pour la contrainte sur les régions



Modèle du problème de Sudoku sous MiniZinc

```
include “alldiferent.mzn”;

% domaine: ensemble d’entier
set of int: domaine = 1..9;

% grille est un tableau à 2 dimensions de cases, indexé par le numéro de ligne et colonne, avec des variables de type integer
array [domaine, domaine] of int: grille;

% grille du sudoku, les 0 correspondant aux cases vides à remplir
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
|];
```

```
% puzzle: variable de décision
% domaine d’association : 1..9
array [domaine, domaine] of var int: puzzle;

% Copie la grille dans un tableau de variables
constraint forall (i, j in domaine) (
        If grille [i, j] > 0 then puzzle [i, j] = grille [i, j] else true endif );

% ligne_diff (ligne) contraint les cases de la ligne ligne à être distinctes
predicate ligne_diff (int: ligne) =
        alldifferent (colonne in domaine) (puzzle[ligne, colonne]);

% colonne_diff(colonne) contraint les cases de la colonne colonne à être distinctes.
predicate colonne_diff (int: colonne) =
        alldifferent (ligne in domaine) (puzzle[ligne, colonne]);
```

Modèle du problème de Sudoku sous MiniZinc

```
% region_diff(ligne) de même pour les régions à être différentes
predicate region_diff (int: ligne, int: colonne) =
        alldifferent (i, j in 0..2) (puzzle[ligne + i, colonne + j]);

% Utilisation des prédicats pour contraindre les valeurs de la grille
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));


% Problème de satisfaction : trouver les valeurs pour les variables
de décisions pour satisfaire les contraintes
solve satisfy;

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

```
% region_irr1(ligne, colonne, a) pour les régions irrégulières.
int: tmp;
predicate region_irr1(int: ligne, int: colonne, int: a) =
        if a == 0 then
                alldifferent(i,j in 0..2) (puzzle[ligne+i, colonne+j])
        else
                alldifferent(i,j in 0..2)
                (if ligne+i == 3 \/ ligne+i == 6 then
                        (puzzle[ligne+i, colonne+j])
                        /\ (puzzle[ligne+i+a, colonne+j])
                        /\ tmp = colonne+j
                elseif colonne+j != tmp then
                        (puzzle[ligne+i+a, colonne+j-a])
                elseif ligne+i == 4 \/ ligne+i == 7 then
                        (puzzle[ligne+i, colonne+j])
                        /\ (puzzle[ligne+i-a, colonne+j])
                        /\ tmp = colonne+j
                elseif colonne+j != tmp then
                        (puzzle[ligne+i-a, colonne+j+a])
                else
                        true
                endif)
        endif;
```

Questions ?

Merci !
Pour nous contacter:

david.toty@etu.upmc.fr

maxime.tran@etu.upmc.fr

Encadrant du projet:

Responsable: Fabrice KORDON
Client: Tarek Menouer