



Rapport Tp sudoku

Meryam RHADI
Lamyae KHAIROUN

9 octobre 2022

Master 2
Génie logiciel
UE IA pour le génie logiciel

Responsable
Nadjib LAZAAR

Sommaire

Titre	1
Sommaire	2
1 Exécution du projet	3
2 Partie TD	3
2.1 Question 1	3
2.2 Question 2	3
2.3 Question 3	3
2.4 Question 4	5
2.5 Question 5	6
3 Partie TP	8
3.1 Question 6	8
3.2 Question 7	8
3.3 Question 8	9
3.4 Question 9	11
4 Modifications en PPC	12
4.1 Question 10	12
4.2 Question 11	13
5 Greater Than Sudoku	14
5.1 Question 12	14

1 Exécution du projet

Le dépôt du projet contient un fichier zippé sous le nom de **SudokuA.jar**, il faut le décompresser, ouvrir une invite de commande dans ce fichier, et exécuter la commande : **java -jar SudokuPPC.jar**

2 Partie TD

2.1 Question 1

La modélisation du problème sudoku($n \times n$) sous la forme d'un réseau de contraintes $N \langle X, D, C \rangle$:

- Variables : $X = X_{1,1} + X_{1,2} + \dots + X_{n,n}$
- Domaine : $D = \{1, \dots, n\}$
- Contraintes : $C =$
 - $\text{allDifferent}(X_{i,1}, \dots, X_{i,n}), \forall i(\text{rows})$
 - $\text{allDifferent}(X_{i,1}, \dots, X_{i,n}), \forall i(\text{columns})$
 - $\text{allDifferent}(X_{i,j}), \forall i, j(\text{squares})$

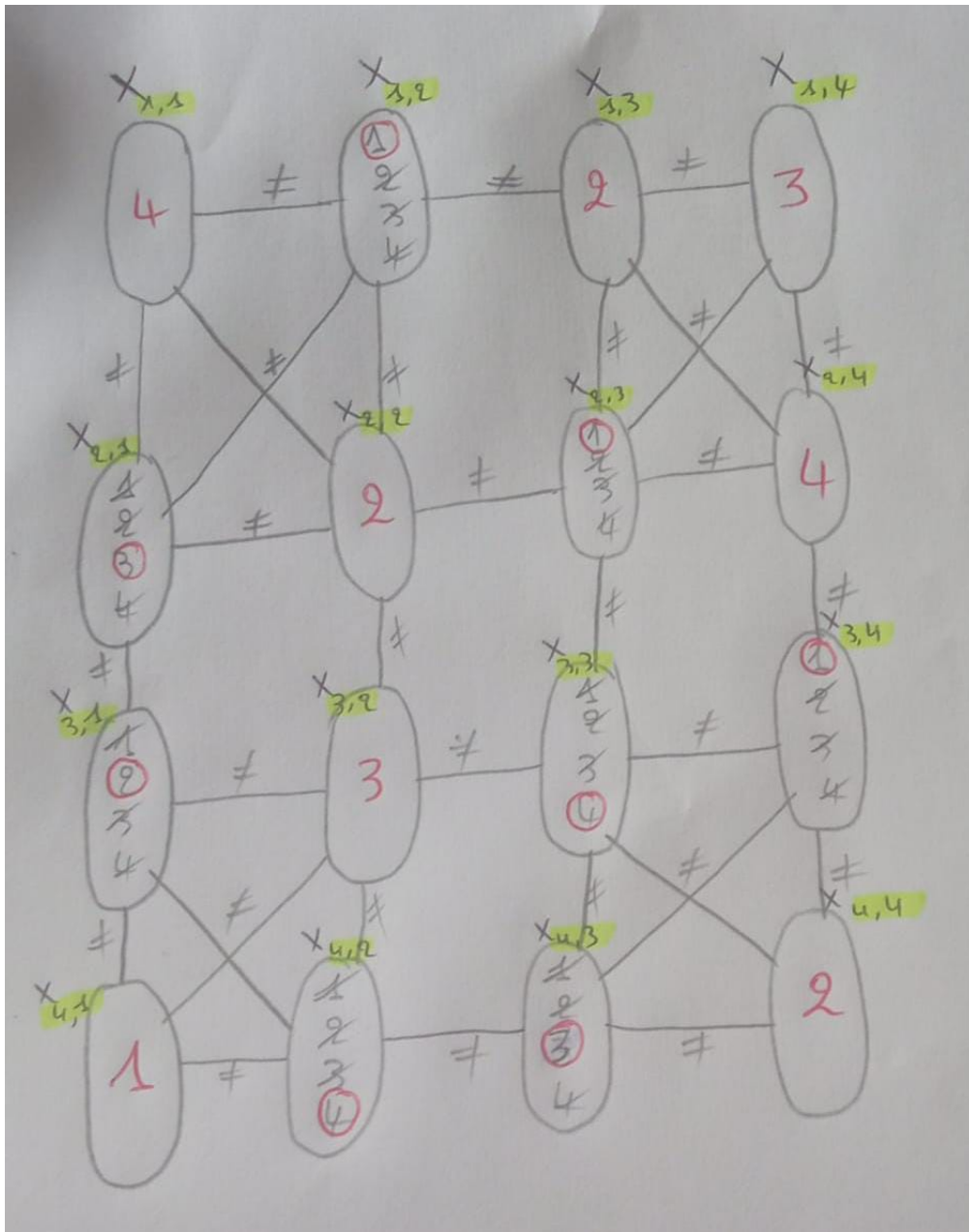
2.2 Question 2

La taille de l'espace de recherche est entre 1 et 4 .

2.3 Question 3

Déroulement l'algorithme du backtrack sur l'instanciation partielle de la figure 1.

- $(X_{1,2}, val = 4) \rightarrow \neq_{1,2} 1,2 \cdot (X_{3,4}, val = 3) \rightarrow \neq_{3,4} 4,3$
- $(X_{1,2}, val = 2) \rightarrow \neq_{1,2} 2,2 \cdot (X_{3,4}, val = 3) \rightarrow \neq_{3,4} 4,3$
- $(X_{2,1}, val = 2) \rightarrow \neq_{2,1} 2,2 \cdot (X_{4,3}, val = 1) \rightarrow \neq_{4,3} 4,1$
- $(X_{2,1}, val = 4) \rightarrow \neq_{2,1} 1,1 \cdot (X_{4,2}, val = 2) \rightarrow \neq_{4,2} 3,1$
- $(X_{3,1}, val = 3) \rightarrow \neq_{3,1} 3,2 \cdot (X_{3,1}, val = 4) \rightarrow \neq_{3,1} 1,1$
- $(X_{3,1}, val = 1) \rightarrow \neq_{3,1} 4,1 \cdot (X_{2,1}, val = 1) \rightarrow \neq_{2,1} 1,2$
- $(X_{4,2}, val = 3) \rightarrow \neq_{4,2} 3,2 \cdot (X_{1,2}, val = 3) \rightarrow \neq_{1,2} 1,4$
- $(X_{4,2}, val = 1) \rightarrow \neq_{4,2} 4,1 \cdot (X_{4,3}, val = 4) \rightarrow \neq_{4,3} 3,3$
- $(X_{4,3}, val = 2) \rightarrow \neq_{4,3} 4,4 \cdot (X_{3,3}, val = 1) \rightarrow \neq_{3,3} 2,3$
- $(X_{3,4}, val = 2) \rightarrow \neq_{3,4} 4,4 \cdot (X_{2,3}, val = 4) \rightarrow \neq_{2,3} 2,4$
- $(X_{3,4}, val = 4) \rightarrow \neq_{3,4} 2,4 \cdot (X_{2,3}, val = 3) \rightarrow \neq_{2,3} 1,4$
- $(X_{3,3}, val = 2) \rightarrow \neq_{3,3} 4,4 \cdot (X_{2,3}, val = 2) \rightarrow \neq_{2,3} 1,3$
- $(X_{3,3}, val = 3) \rightarrow \neq_{3,3} 3,2$

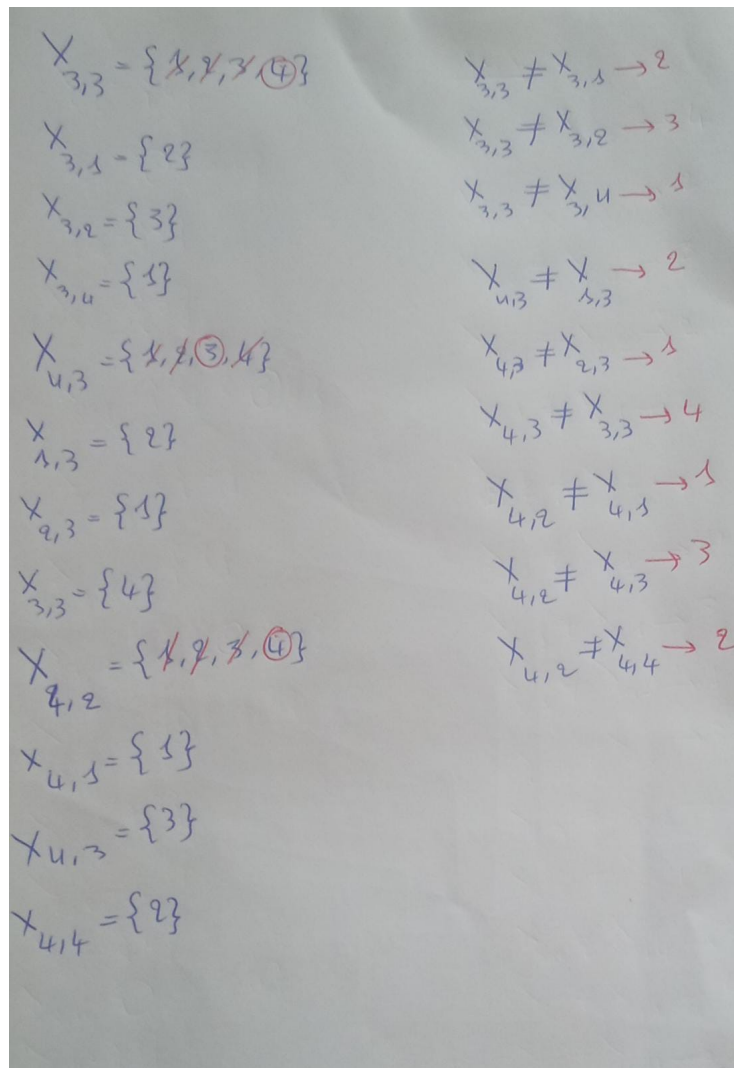


2.4 Question 4

$X_{1,1} = \{4\}$
 $X_{1,2} = \{\textcircled{1}, 2, 3, 4\}$
 $X_{1,3} = \{2\}$
 $X_{1,4} = \{3\}$
 $X_{2,1} = \{\cancel{1}, \cancel{2}, \textcircled{3}, 4\}$
 $X_{2,2} = \{2\}$
 $X_{3,1} = \{\cancel{1}, \textcircled{2}, \cancel{3}, 4\}$
 $X_{2,3} = \{\textcircled{1}, \cancel{2}, \cancel{3}, 4\}$
 $X_{2,4} = \{3\}$
 $X_{2,2} = \{2\}$
 $X_{2,4} = \{4\}$
 $X_{3,4} = \{\textcircled{1}, \cancel{2}, \cancel{3}, 4\}$
 $X_{1,4} = \{3\}$
 $X_{2,4} = \{4\}$
 $X_{4,4} = \{2\}$

Ans:

$X_{1,2} \neq X_{1,1} \rightarrow 4$
 $X_{1,2} \neq X_{1,3} \rightarrow 2$
 $X_{1,2} \neq X_{1,4} \rightarrow 3$
 $X_{2,1} \neq X_{2,2} \rightarrow 2$
 $X_{2,1} \neq X_{1,1} \rightarrow 4$
 $X_{2,1} \neq X_{1,2} \rightarrow 1$
 $X_{3,1} \neq X_{1,1} \rightarrow 4$
 $X_{3,1} \neq X_{2,1} \rightarrow 3$
 $X_{3,1} \neq X_{4,1} \rightarrow 1$
 $X_{2,3} \neq X_{2,1} \rightarrow 3$
 $X_{2,3} \neq X_{2,2} \rightarrow 2$
 $X_{2,3} \neq X_{2,4} \rightarrow 4$
 $X_{3,4} \neq X_{1,4} \rightarrow 3$
 $X_{3,4} \neq X_{2,4} \rightarrow 4$
 $X_{3,4} \neq X_{4,4} \rightarrow 2$



2.5 Question 5

Pour que l'algorithme de **Backtrack 1** retourne l'ensemble des solutions, il suffit de changer le `true` qui est dans l'algorithme initial de **Backtrack** par `false`.

Comme il est indiqué dans l'algorithme **2**, qui est la version modifiée qui permet de trouver tous les solutions possibles pour un sudoku donné.

Algorithme 1. Algo qui retourne une seule solution

```
1 Backtrack(<X,D,C>, I) :  
2 if I is complete then  
3   | return true;  
4 end if  
5 Select a variable  $X_i$  not in I  
6 foreach  $v$  in  $D(X_i)$  do  
7   | if I union  $\langle X_i, v \rangle$  is locally consistent then  
8     |   | if Backtrack(< X,D,C > ,I union  $\langle X_i, v \rangle$ ) then  
9       |   |   | return true  
10    |   end if  
11 end foreach  
12 return false
```

Algorithme 2. Algo qui retourne tous les solutions possible

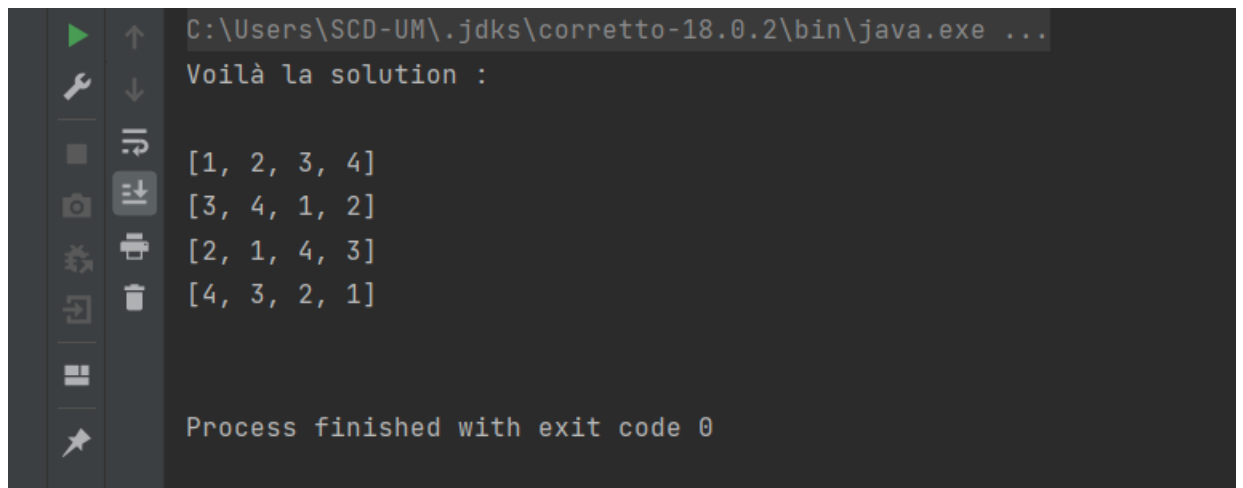
```
1 Backtrack(<X,D,C>, I) :  
2 if I is complete then  
3   | return true;  
4 end if  
5 Select a variable  $X_i$  not in I  
6 foreach  $v$  in  $D(X_i)$  do  
7   | if I union  $\langle X_i, v \rangle$  is locally consistent then  
8     |   | if Backtrack(< X,D,C > ,I union  $\langle X_i, v \rangle$ ) then  
9       |   |   | return false  
10    |   end if  
11 end foreach  
12 return false
```

3 Partie TP

3.1 Question 6

Lien vers le code sur github :

Le résultat de l'exécution d'un sudoku de taille 4 x 4, en utilisant le back-tracking :

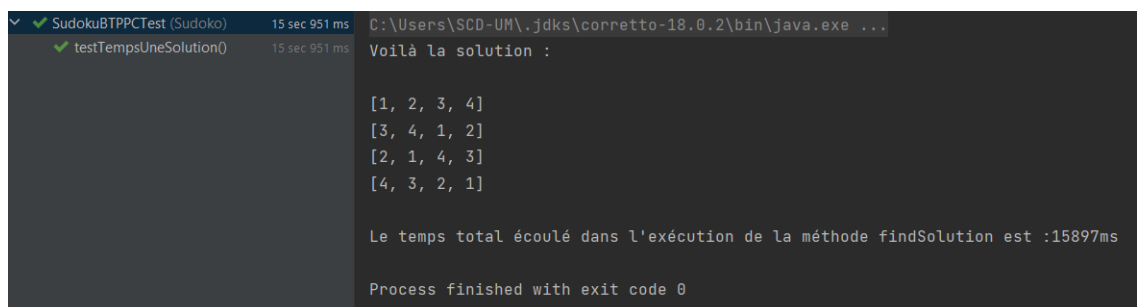


```
C:\Users\SCD-UM\.jdk\corretto-18.0.2\bin\java.exe ...  
Voilà la solution :  
  
[1, 2, 3, 4]  
[3, 4, 1, 2]  
[2, 1, 4, 3]  
[4, 3, 2, 1]  
  
Process finished with exit code 0
```

3.2 Question 7

Pour comparer les deux méthodes de la résolution du sudoku, on a pensé à faire un test unitaire qui permet de calculer le temps total écoulé dans l'exécution de la méthode qui permet de trouver une solution du sudoku.

Pour le modèle de BT, en effectuant le test pour un sudoku 4x4 le résultat était de 15s, comme montre la figure 1.



```
✓ SudokuBTPPCTest (Sudoku) 15 sec 951 ms  
✓ testTempsUneSolution() 15 sec 951 ms  
Voilà la solution :  
  
[1, 2, 3, 4]  
[3, 4, 1, 2]  
[2, 1, 4, 3]  
[4, 3, 2, 1]  
  
Le temps total écoulé dans l'exécution de la méthode findSolution est :15897ms  
  
Process finished with exit code 0
```

Figure 1. Résultat du test pour BT

Et ont effectuant le même test pour un sudoku de taille 4x4, en utilisant le modèle PPC, on remarque une grande différence au niveau du temps total écoulé dans l'exécution, comme montre la figure 2.

```
- Complete search - 1 solution found.  
Model[Sudoku problem]  
Solutions: 1  
Building time : 0,205s  
Resolution time : 0,095s  
Nodes: 8 (83,9 n/s)  
Backtracks: 0  
Fails: 0  
Restarts: 0
```

Figure 2. Résultat du test pour PPC

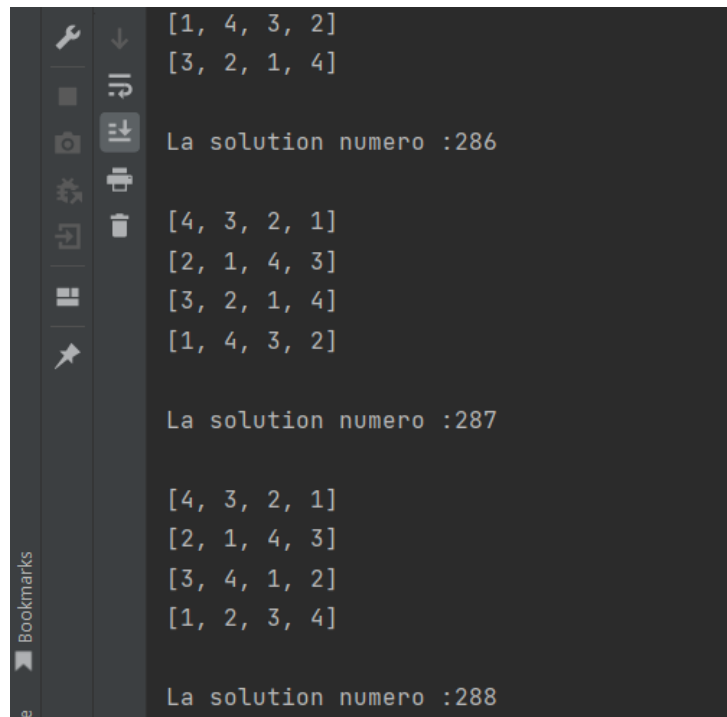
Donc comme conclusion, on peut dire que La résolution par choco-solver est plus rapide et efficace.

3.3 Question 8

Pour la modification des deux algorithmes pour qu'ils puissent retourner l'ensemble de toutes les solutions du sudoku, au lieu d'une seule. Pour le Backtracking, on a juste modifié le true, qui permettre de sortir de la boucle quand il trouve une solution, par false pour qu'il continue la recherche des solutions, jusqu'à qu'il trouve plus, comme montre la figure 3, et la figure 4 montre un extrait du résultat de l'exécution sur un sudoku de taille 4x4.

```
public boolean findSolutionAll(int i, int j) {  
    int[] update;  
    if (i == -1 && j == -1) {  
        if (solutionChecker()) {  
            compteur++;  
            System.out.println(this);  
            System.out.println("La solution numero : " + compteur);  
            return false;  
        } else {  
            return false;  
        }  
    }  
}
```

Figure 3. Code pour modifier le backtracking



```
[1, 4, 3, 2]
[3, 2, 1, 4]

La solution numero :286

[4, 3, 2, 1]
[2, 1, 4, 3]
[3, 2, 1, 4]
[1, 4, 3, 2]

La solution numero :287

[4, 3, 2, 1]
[2, 1, 4, 3]
[3, 4, 1, 2]
[1, 2, 3, 4]

La solution numero :288
```

Figure 4. Extrait du résultat de tous les solutions avec BT

Et pour le ppc avec choco solver, on a ajouter une boucle while qui permette de chercher la solution tant qu'il existe, la figure 5 montre le code qui permettre de retourner tous les solutions, et la figure 6 montre un extrait du résultat de l'exécution sur un sudoku de taille 4x4.

```
public void solveAll() {
    buildModel();
    while (model.getSolver().solve()) {
        model.getSolver().solve();
        printGrid();
    }
    model.getSolver().printStatistics();
}
```

Figure 5. Code pour modifier PPC

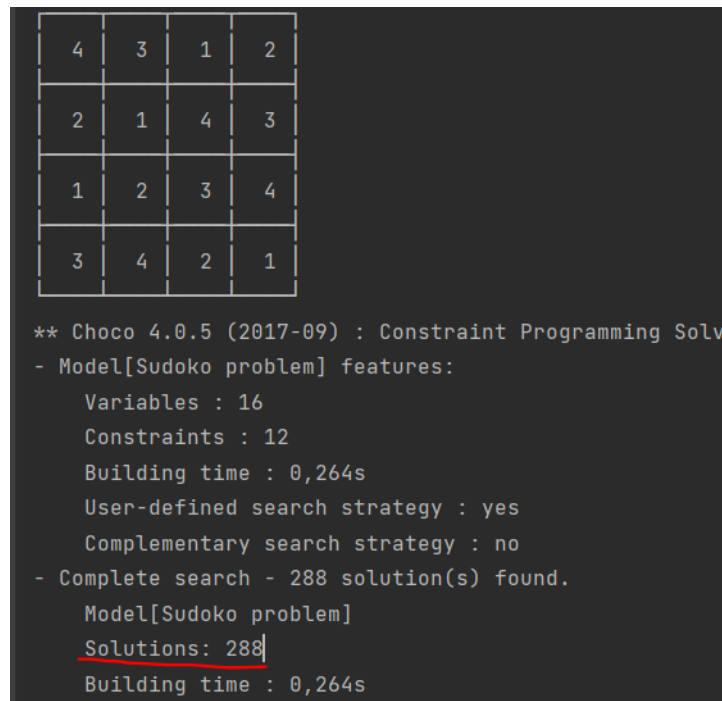


Figure 6. Extrait du résultat de tous les solutions avec PPC

3.4 Question 9

Comme dans la question 7, même dans cette question on a pensé à faire un test unitaire qui permet de calculer le temps d'exécution de la méthode qui permet de trouver toutes les solutions du sudoku dans BT et PPC.

Pour le modèle de BT, en effectuant le test pour un sudoku 4x4 le résultat était 195,376s ce qui donne 3,25626667 minute pour trouver tous les solutions possibles, comme montre la figure 7.

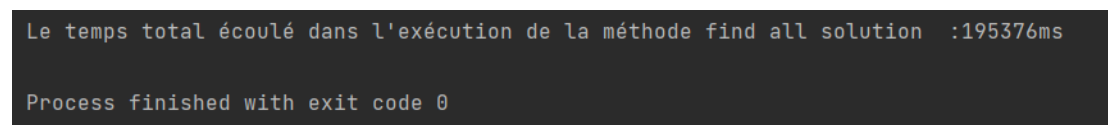


Figure 7. Résultat du test pour BT

Et ont effectuant le même test pour trouver tous les solutions possibles pour un sudoku de taille 4x4, en utilisant le modèle PPC, on remarque également la grande différence au niveau du temps total écoulé dans l'exécution, comme montre la figure 8.

```
Model[Sudoku problem]
Solutions: 288
Building time : 0,430s
Resolution time : 0,334s
Nodes: 577 (1726,3 n/s)
Backtracks: 579
Fails: 2
Restarts: 0
```

Figure 8. Résultat du test pour PPC

On remarque une très grande différence au niveau du temps de l'exécution demandé pour trouver toutes les solutions possibles, pour le BT et PPC.

4 Modifications en PPC

4.1 Question 10

Pour répondre à cette question, on a utilisé la bibliothèque CSV pour remplir la matrice par les instances proposées dans la figure du Td pour le sudoku de taille 9x9, comme montre la figure 9.

```
public int[][] load() {
    try {
        int[][] matrix = Files.lines(Paths.get( first: "src/main/java/Fichier/mat
        .map(line -> Arrays.stream(line.split( regex: ";" )) Stream<String>
        .mapToInt(Integer::parseInt) IntStream
        .toArray() Stream<int[]>
        .toArray(int[][]::new);
        return matrix;
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Figure 9. Importation de fichier csv

Et après on a donnée cette matrice au Model pour trouver la ou les solutions possibles pour ce sudoku prédéfini.

```
case 9: {
    int valeur = matricePredefini[i][j];
    if (valeur < 1) {
        rows[i][j] = model.intVar(format("[%s.%s]", i, j), lb: 1, ub: 9);
        cols[j][i] = rows[i][j];
    } else {
        rows[i][j] = model.intVar(matricePredefini[i][j]);
        cols[j][i] = rows[i][j];
    }
    break;
}
```

4.2 Question 11

Pour répondre à cette question, on a utilisé également la bibliothèque **CSV**, pour remplir la matrice par les instances proposées dans la figure du TD, pour le sudoku de taille 16x16.

Et après, on a modifié dans les conditions de intVar, pour quel prendre en compte le sudoku de taille 16, et on a utilisé une hashmap pour stocker les lettres de A à F, et les modifier dans l'affichage, la figure [16] montre le résultat de l'exécution.

```

----- Jeu de sudoku -----
Bienvenue dans le jeu de sudoku, Merci de rentré la taille de la grille, qui doit etre de taille
Enter un nombre:
16
Solution pour sudoku 16*16:

```

A	G	E	1	F	8	9	6	4	B	D	5	7	2	3	C
6	C	8	F	1	B	4	E	2	7	3	A	D	G	5	9
B	9	5	D	2	3	G	7	F	E	C	1	6	4	8	A
2	7	4	3	A	C	D	5	9	8	G	6	1	B	F	E
7	4	6	5	8	F	2	3	A	1	B	E	9	D	C	G
8	F	G	A	9	7	C	4	D	6	5	3	B	1	E	2
C	1	3	E	D	A	6	B	7	G	9	2	8	F	4	5
9	D	B	2	E	G	5	1	8	4	F	C	3	7	A	6
G	B	A	8	4	2	1	C	3	9	7	F	E	5	6	D
5	6	F	9	3	D	A	G	E	2	1	B	C	8	7	4
D	2	1	4	6	E	7	9	5	C	8	G	A	3	B	F
3	E	7	C	B	5	F	8	6	D	A	4	G	9	2	1

Figure 10. Résultat du sudoku 16

5 Greater Than Sudoku

5.1 Question 12

Pour répondre à cette question, on a créé deux fichiers .csv, un qui contient les signes de comparaison qui sont dans les lignes, et l'autre celle des colonnes, les deux fichiers sont stockés chacune dans une matrice. On a juste adapté la taille de la matrice, pour que les deux matrices vont avoir la même taille, dans le but de simplifier le traitement.

Et après pour faire le traitement, on a utilisé **arithm** entre les cases, et on donne comme opération les éléments des deux matrices de signes, on obtient comme résultat la matrice suivante :

- Tapez 3 si vous voulez voir le résultat de sudoku Greater Than Sudoku

3

Solution pour sudoku 9*9:

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