

Sudoku solver with OR-tools



Google OR-Tools

Edouard Botherel – Valentin Davis - Jean-Benoit Troude - Geoffroi Villamaux

Projet : Sudoku x OR-Tools



1. L'aide de la librairie google OR-tools.
2. Intégration de l'algorithme dans le projet à l'aide github et visual studio community.
3. Utilisation de l'algorithme depuis la console pour chaque étudiant

Index

I. Organisation du Projet

- A. Phases du projet
- B. Teamwork

II. Présentation d'OR-Tools

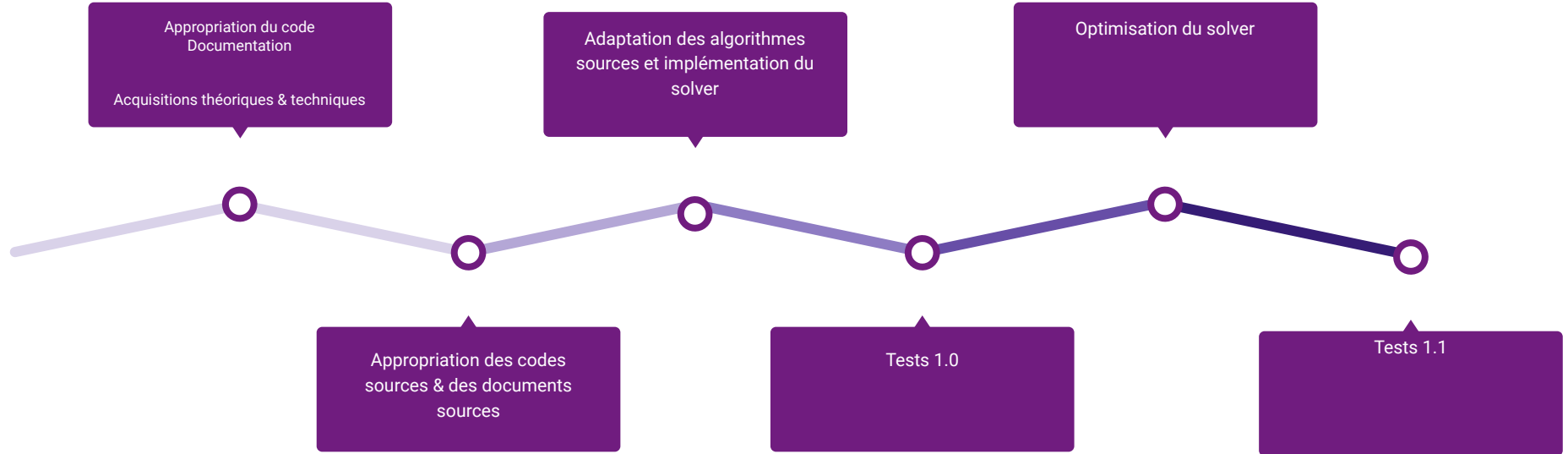
- A. Description de la librairie
- B. Exemples de fonctionnement

III. Algorithme & Code

- A. Classes Benchmark & Core
 - B. Fonctionnement du code
 - C. Résultat
-

I - Organisation du Projet

Phases du Projet

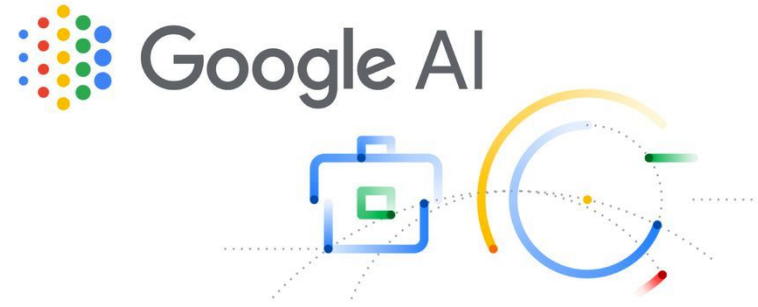


Teamwork

		Théorie	Algorithmie	Développement	Diaporama
1	Jean-Benoit Troude		x	X	x
2	Edouard Botherel	x	X	x	
3	Geoffroi Villamaux	x	x		X
4	Valentin Davis	X	x	x	

II - Présentation d'OR-Tools

OR - TOOLS ?



OR-Tools est une librairie d'optimisation à code source ouvert, conçue pour résoudre les problèmes concernant le routage des véhicules, de flux, de programmation linéaire et en nombres entiers et de programmation par contraintes.



Exemple de fonctionnement : “The N-queens Problem”

Comment peut-on placer N reines sur un échiquier $N \times N$ de sorte que deux d'entre elles ne s'attaquent pas l'une l'autre ?

Sachant qu'aux échecs, une dame peut attaquer horizontalement, verticalement et diagonalement.

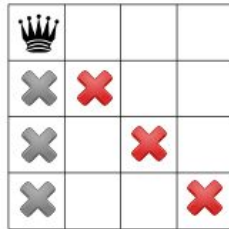
Le solveur OR - TOOLS permet d'essayer toutes les affectations possibles afin de déterminer toutes les solutions possibles.

Pour ce faire, on utilise :

- La propagation
- Le retour en arrière

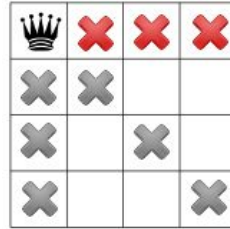


La propagation



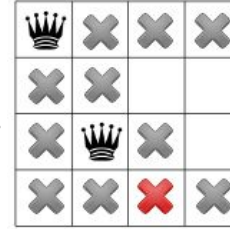
Sous-étape 1

On place la première reine



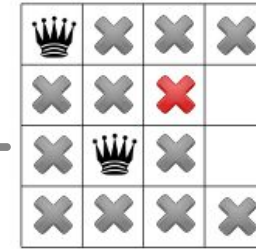
Sous-étape 2

On élimine les emplacements impossibles



Sous-étape 3

On place la seconde reine



Sous-étape 4

On élimine les emplacements impossibles

Le retour en arrière

×	×		
♔	×	×	×
×	×		
×	♔	×	

Sous-étape 1

On place la première reine



×	×	♔	
♔	×	×	×
×	×	×	
×	♔	×	

Sous-étape 2

On place les reines suivantes



×	×	♔	×
♔	×	×	×
×	×	×	♔
×	♔	×	×

Sous-étape 3

Solution au problème



III - Algorithmme & Code

Classe Benchmark

Comme son nom l'indique le Benchmark permet de comparer la performance de chaque méthode de résolution en comparant le temps d'exécution en millisecondes.

```
public void Benchmark()
{
    foreach (var puzzle in IterationPuzzles)
    {
        Console.WriteLine($"Solver {SolverPresenter} solving sudoku: \n {puzzle}");
        var startTime = Clock.Elapsed;
        var solution = SolverPresenter.SolveWithTimeLimit( puzzle, MaxSolverDuration);
        if (!solution.IsValid(puzzle))
        {
            throw new ApplicationException($"sudoku has {solution.NbErrors(puzzle)} errors");
        }

        var duration = Clock.Elapsed - startTime;
        var durationSeconds = (int) duration.TotalSeconds;
        var durationMilliseconds = duration.TotalMilliseconds - (1000 * durationSeconds);
        Console.WriteLine($"Valid Solution found: \n {solution} \n Solver {SolverPresenter} found the solution in {durationSeconds}
    }
}
```

Classe Core

La classe Core inclut les grilles de sudoku. Une liste de 81 int est déclarée pour les cellules. La grille est construite de manière à avoir un accès facile aux index des lignes et colonnes et permet de comparer les cases voisines.

Elle inclut les méthodes permettant de tester les valeurs proposées par le solveur pour résoudre le sudoku et les test afin de valider la solution

```
static GrilleSudoku()[...]
```

```
/// <summary> constructor that initializes the board with 81 cells
```

1 référence

```
public GrilleSudoku(IEnumerable<int> cells)[...]
```

1 référence

```
public GrilleSudoku()[...]
```

```
// The List property makes it easier to manipulate cells,
```

14 références

```
public List<int> Cellules { get; set; } = Enumerable.Repeat(0, 81).ToList();
```

```
/// <summary> Easy access by row and column number
```

10 références

```
public int GetCellule(int x, int y)[...]
```

```
/// <summary> Easy setter by row and column number
```

3 références

```
public void SetCell(int x, int y, int value)[...]
```

```
/// <summary> Displays a GrilleSudoku in an easy-to-read format
```

8 références

0 références

```
public int[] GetPossibilities(int x, int y)...
```

```
/// <summary> Parses a single GrilleSudoku
```

0 références

```
public static GrilleSudoku Parse(string sudokuAsString)...
```

```
/// <summary> Parses a file with one or several sudokus
```

1 référence

```
public static List<GrilleSudoku> ParseFile(string fileName)...
```

```
/// <summary> Parses a list of lines into a list of sudoku, accounting for most ...
```

2 références

```
public static List<GrilleSudoku> ParseMulti(string[] lines)...
```

```
/// <summary> identifies characters to be parsed into sudoku cells
```

2 références

```
private static bool IsSudokuChar(char c)...
```

0 références

```
public object Clone()...
```

4 références

```
public Core.GrilleSudoku CloneSudoku()...
```

Code V1.0

Initialisation des paramètres

```
int cell_size=3;  
  
IEnumerable<int> CELL = Enumerable.Range(0, 3);  
  
IEnumerable<int> cellIndices = Enumerable.Range(0, 9);
```

Variables de contrainte

```
IntVar[,] grid = solver.MakeIntVarMatrix(9, 9, 1, 9, "grid");  
IntVar[] grid_flat = grid.Flatten();
```

Création de la grille



Code V1.0

Ajout des contraintes à partir de la grille créée

```
//Masque de résolution
for (int i=0; i < cellIndices.Count(); i++)
{
    for (int j=0; j < cellIndices.Count(); j++)
    {
        if (s.GetCellule(i, j) > 0)
        {
            solver.Add(grid[i, j] == s.GetCellule(i, j));
        }
    }
}

//Un chiffre ne figure qu'une seule fois par ligne/colonne/cellule
for (int i=0; i < cellIndices.Count(); i++)
{
    // Lignes
    solver.Add((from j in cellIndices
                select grid[i, j]).ToArray().AllDifferent());

    // Colonnes
    solver.Add((from j in cellIndices
                select grid[j, i]).ToArray().AllDifferent());
}

//Cellules
for (int i=0; i < CELL.Count(); i++)
{
    for (int j = 0; j < CELL.Count(); j++)
    {
        solver.Add((from di in CELL
                    from dj in CELL
                    select grid[i * cell_size + di, j * cell_size + dj]
                    ).ToArray().AllDifferent());
    }
}
```

Ajout des contraintes à respecter au solveur lors de la résolution

Ajout des lignes et colonnes au solveur.

Toutes les valeurs doivent être différentes.

Ajout des cellules du sudoku au solveur

Vérification des dimensions.

Code V1.0

Résolution du sudoku

```
//Début de la résolution
DecisionBuilder db = solver.MakePhase(grid_flat,
    Solver.INT_VAR_SIMPLE,
    Solver.INT_VALUE_SIMPLE);
solver.NewSearch(db);

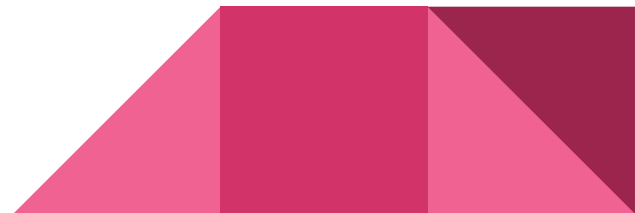
//Mise à jour du sudoku
//int n = cell_size * cell_size;
//Or on sait que cell_size = 3 -> voir ligne 13
//Inspiré de l'exemple : taille des cellules identique
while (solver.NextSolution())
{
    for (int i = 0; i < 9; i++)
    {
        for (int j = 0; j < 9; j++)
        {
            s.SetCell(i,j, (int) grid[i,j].Value());
        }
    }
}

solver.EndSearch();
```

Création de la grille dans la console.

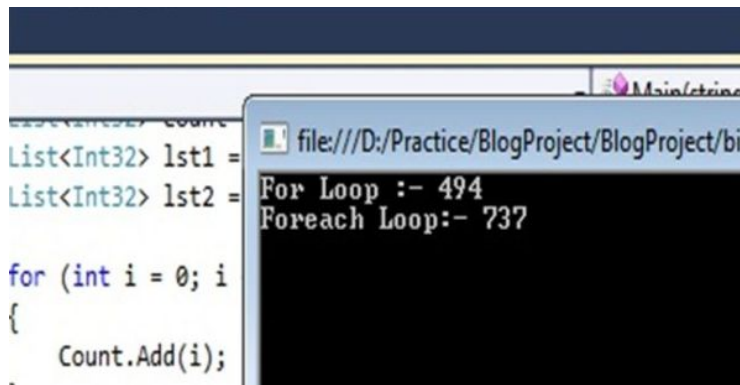
Début de la recherche de solution

Filtrage des différentes solutions
et intégration des plus pertinentes



Optimisation

Recherche et travail réalisé



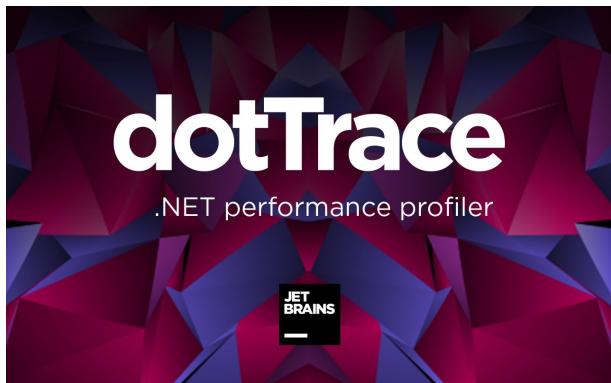
The screenshot shows a code editor with the following C# code:

```
List<Int32> lst1 =  
List<Int32> lst2 =  
  
for (int i = 0; i  
{  
    Count.Add(i);  
.
```

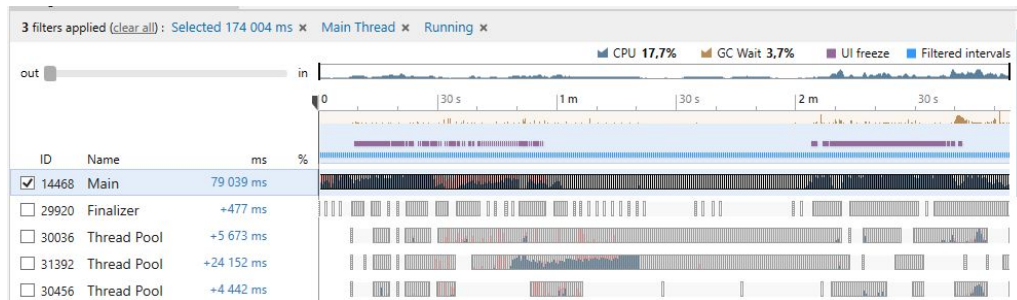
Overlaid on the code is a window titled "file:///D:/Practice/BlogProject/BlogProject/bi" showing performance results:

```
For Loop :- 494  
Foreach Loop:- 737
```

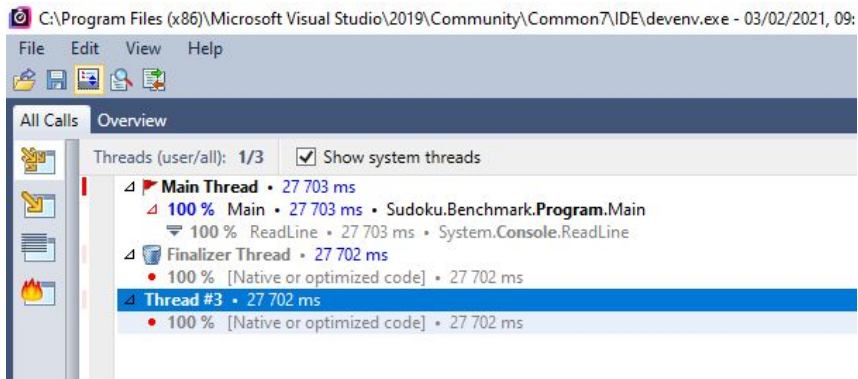
- Remplacer boucle foreach par for
- Tentative d'utilisation des structures à la place des classes
- Réflexion sur la méthode `grid.Flatten()`
- Utilisation du profiler `dotTrace/ReSharper`



Optimisation JetBrain profiler



Titre : *Puissance de calcul nécessaire à la compilation*



Titre : *Temps nécessaire à la compilation*

Résultats

```
Choose a solver:
1 - Sudoku.Core.EmptySolver
2 - Sudoku.DL.DLSolver
3 - Sudoku.DL.OR.Sudoku_OR
4 - Sudoku.Probabilistic.ProbabilisticSolver
5 - Sudoku.SolverHumain.SolverHumain
3
Valid solution:
-----
| 3 5 1 | 2 8 6 | 4 9 7 |
| 4 9 2 | 1 5 7 | 6 3 8 |
| 7 8 6 | 9 3 4 | 5 1 2 |
-----
| 2 7 5 | 4 6 9 | 1 8 3 |
| 9 3 8 | 5 2 1 | 7 6 4 |
| 6 1 4 | 8 7 3 | 2 5 9 |
-----
| 8 2 9 | 6 4 5 | 3 7 1 |
| 1 6 3 | 7 9 2 | 8 4 5 |
| 5 4 7 | 3 1 8 | 9 2 6 |
-----

Time to solution: 154,043 ms
Select Mode:
1-Single Solver Test,
2-Complete Benchmark (40 s max per sudoku),
3-Complete Benchmark (5 mn max per GrilleSudoku),
4-Exit program
```

Titre : Résultat sudoku 1 facile avant optimisation

154.03ms

```
Choose a solver:
1 - Sudoku.Core.EmptySolver
2 - Sudoku.DL.DLSolver
3 - Sudoku.DL.OR.Sudoku_OR
4 - Sudoku.Probabilistic.ProbabilisticSolver
5 - Sudoku.SolverHumain.SolverHumain
3
Valid solution:
-----
| 4 8 3 | 9 2 1 | 6 5 7 |
| 9 6 7 | 3 4 5 | 8 2 1 |
| 2 5 1 | 8 7 6 | 4 9 3 |
-----
| 5 4 8 | 1 3 2 | 9 7 6 |
| 7 2 9 | 5 6 4 | 1 3 8 |
| 1 3 6 | 7 9 8 | 2 4 5 |
-----
| 3 7 2 | 6 8 9 | 5 1 4 |
| 8 1 4 | 2 5 3 | 7 6 9 |
| 6 9 5 | 4 1 7 | 3 8 2 |
-----

Time to solution: 2,939 ms
```

Titre : Résultat sudoku 1 facile après optimisation

2.939 ms



Merci de votre attention !