Sudoku Solver: Dancing Links

Cynthia Wang, Valentin Guisnet et Louis Loisel

Sommaire

01

Partie théorique

- Hierarchie de Karp
- Couverture exacte
- Donald Knuth
- Algorithme X

02

Partie technique

- Le benchmark
- La librairie: DlxLib
- Notre code

03

Gestion du projet

- Notre démarche
- Les difficultés
- GitHub
- Les résultats

Partie théorique

01

Hierarchie de Karp

Les 21 problèmes NP-complets de Karp ont marqué une étape importante de l'histoire de la théorie de la complexité des algorithmes. Ce sont 21 problèmes réputés difficiles de combinatoire et de théorie des graphes qui sont réductibles entre eux.

En résolvant un sudoku à l’aide de la méthode des Dancing Links nous nous basons sur le problème de la couverture exacte.

Problème de la couverture exacte

Soit $U = \{0, 1, 2, 3, 4\}$ et soit $\mathcal{S} = \{\mathbf{E}, \mathbf{I}, \mathbf{P}\}$ une collection de trois ensembles :

- $\mathbf{E} = \{0, 2, 4\}$;
- $\mathbf{I} = \{1, 3\}$;
- $\mathbf{P} = \{2, 3\}$.

Exemple 1

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

	1	2	3	4	5	6	7
B	1	0	0	1	0	0	0
D	0	0	1	0	1	1	0
F	0	1	0	0	0	0	1

Exemple 2

Problème de la couverture exacte dans le cas des sudoku

4 types de contraintes :

- Ligne-Colonne
- Ligne-Nombre
- Colonne-Nombre
- Boite-Nombre

9*9*9=729 Possibilités

81+81+81+81=324 Contraintes

Contraintes Ligne-Colonne

	R1 C1	R1 C2	...
R1C1#1	1	0	...
R1C1#2	1	0	...
R1C1#3	1	0	...
R1C1#4	1	0	...
R1C1#5	1	0	...
R1C1#6	1	0	...
R1C1#7	1	0	...
R1C1#8	1	0	...
R1C1#9	1	0	...
R1C2#1	0	1	...
R1C2#2	0	1	...
R1C2#3	0	1	...
R1C2#4	0	1	...
R1C2#5	0	1	...
R1C2#6	0	1	...
R1C2#7	0	1	...
R1C2#8	0	1	...
R1C2#9	0	1	...
...	...	...	...

Contraintes Ligne-Nombre

	R1 #1	R1 #2	...
R1C1#1	1	0	...
R1C1#2	0	1	...
...	...	...	...
R1C2#1	1	0	...
R1C2#2	0	1	...
...	...	...	...
R1C3#1	1	0	...
R1C3#2	0	1	...
...	...	...	...
R1C4#1	1	0	...
R1C4#2	0	1	...
...	...	...	...
R1C5#1	1	0	...
R1C5#2	0	1	...
...	...	...	...
R1C6#1	1	0	...
R1C6#2	0	1	...
...	...	...	...
R1C7#1	1	0	...
R1C7#2	0	1	...
...	...	...	...
R1C8#1	1	0	...
R1C8#2	0	1	...
...	...	...	...
R1C9#1	1	0	...
R1C9#2	0	1	...
...	...	...	...

Contraintes Colonne-Nombre

	C1 #1	C1 #2	...
R1C1#1	1	0	...
R1C1#2	0	1	...
...	...	...	...
R2C1#1	1	0	...
R2C1#2	0	1	...
...	...	...	...
R3C1#1	1	0	...
R3C1#2	0	1	...
...	...	...	...
R4C1#1	1	0	...
R4C1#2	0	1	...
...	...	...	...
R5C1#1	1	0	...
R5C1#2	0	1	...
...	...	...	...
R6C1#1	1	0	...
R6C1#2	0	1	...
...	...	...	...
R7C1#1	1	0	...
R7C1#2	0	1	...
...	...	...	...
R8C1#1	1	0	...
R8C1#2	0	1	...
...	...	...	...
R9C1#1	1	0	...
R9C1#2	0	1	...
...	...	...	...

Contraintes Boîte-Nombre

	B1 #1	B1 #2	...
R1C1#1	1	0	...
R1C1#2	0	1	...
...	...	...	...
R1C2#1	1	0	...
R1C2#2	0	1	...
...	...	...	...
R1C3#1	1	0	...
R1C3#2	0	1	...
...	...	...	...
R2C1#1	1	0	...
R2C1#2	0	1	...
...	...	...	...
R2C2#1	1	0	...
R2C2#2	0	1	...
...	...	...	...
R2C3#1	1	0	...
R2C3#2	0	1	...
...	...	...	...
R3C1#1	1	0	...
R3C1#2	0	1	...
...	...	...	...
R3C2#1	1	0	...
R3C2#2	0	1	...
...	...	...	...
R3C3#1	1	0	...
R3C3#2	0	1	...
...	...	...	...

Donald Knuth

- L’un des pionniers de l’algorithmique et a fait de nombreuses contributions dans plusieurs branches de l’informatique théorique
- Auteur de nombreux articles et livres dans le même domaine qui demeurent à l’heure d’aujourd’hui des ouvrages de référence
- Il a reçu plus d’une centaine de prix et d’honneurs dans le monde
- Inventeur de l’algorithme X

Algorithme X

- si la matrice A est vide, le problème est résolu et retourner avec succès;
- choisir une colonne c dans A;
- choisir une ligne r dans la colonne c;
- inclure la ligne r dans la solution partielle;
- pour chaque indice j tel que A[r, j] = 1:
  - supprimer la colonne d'indice j de la matrice A;
  - pour chaque indice i tel que A[i, j] = 1:
    - supprimer la ligne d'indice i de la matrice A;
- répéter l'algorithme récursivement sur la matrice réduite A.

Exemple

Enoncé

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

Considérons un problème de couverture exacte sur l'univers $\mathcal{U} = \{1, 2, 3, 4, 5, 6, 7\}$, avec la collection d'ensembles $\mathcal{S} = \{A, B, C, D, E, F\}$ telle que :

- $A = \{1, 4, 7\}$;
- $B = \{1, 4\}$;
- $C = \{4, 5, 7\}$;
- $D = \{3, 5, 6\}$;
- $E = \{2, 3, 6, 7\}$ et
- $F = \{2, 7\}$.

Niveau 0

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

Niveau 1 : Ligne A

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

	2	3	5	6
D	0	1	1	1

Niveau 1 : Ligne B

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

	2	3	5	6	7
D	0	1	1	1	0
E	1	1	0	1	1
F	1	0	0	0	1

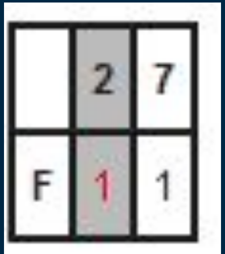
Niveau 2: Ligne D

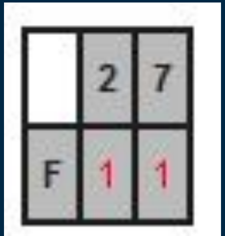
	2	3	5	6	7
D	0	1	1	1	0
E	1	1	0	1	1
F	1	0	0	0	1

	2	3	5	6	7
D	0	1	1	1	0
E	1	1	0	1	1
F	1	0	0	0	1

	2	7
F	1	1

Niveau 3: Ligne F





Partie technique

02

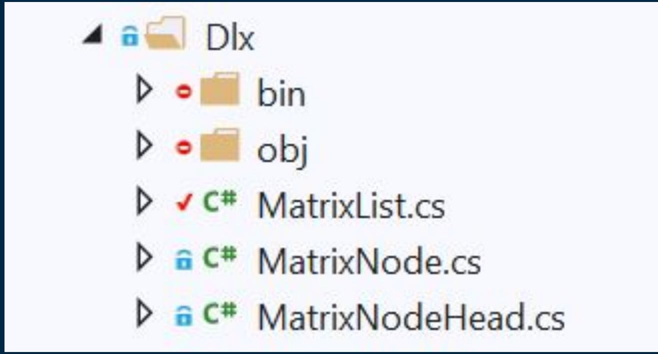
Benchmark

'Benchmarking GrilleSudoku Solvers'	SwarmSolver	Medium	NA	NA	NA	NA	NA	NA	?
'Benchmarking GrilleSudoku Solvers'	SwarmSolver	Hard	NA	NA	NA	NA	NA	NA	?
'Benchmarking GrilleSudoku Solvers'	DLSolver	Hard	46.56 ms	35.24 ms	1.932 ms	44.66 ms	48.52 ms	46.50 ms	1
'Benchmarking GrilleSudoku Solvers'	DLSolver	Easy	49.04 ms	68.76 ms	3.769 ms	46.56 ms	53.38 ms	47.18 ms	2
'Benchmarking GrilleSudoku Solvers'	DLSolver	Medium	49.18 ms	47.37 ms	2.597 ms	46.31 ms	51.37 ms	49.86 ms	2
'Benchmarking GrilleSudoku Solvers'	Sudoku_OR	Easy	51.57 ms	77.17 ms	4.230 ms	47.65 ms	56.05 ms	51.02 ms	3
'Benchmarking GrilleSudoku Solvers'	Sudoku_OR	Medium	53.31 ms	90.86 ms	4.981 ms	47.76 ms	57.39 ms	54.77 ms	4
'Benchmarking GrilleSudoku Solvers'	SolverHumain	Easy	80.25 ms	275.88 ms	15.122 ms	66.74 ms	96.58 ms	77.43 ms	5
'Benchmarking GrilleSudoku Solvers'	Z3Solver	Easy	680.06 ms	128.15 ms	7.024 ms	671.98 ms	684.71 ms	683.49 ms	6
'Benchmarking GrilleSudoku Solvers'	Z3Solver	Medium	1,426.69 ms	1,223.00 ms	67.037 ms	1,380.28 ms	1,503.55 ms	1,396.26 ms	7
'Benchmarking GrilleSudoku Solvers'	Sudoku_OR	Hard	2,199.21 ms	1,145.01 ms	62.762 ms	2,162.35 ms	2,271.68 ms	2,163.60 ms	8
'Benchmarking GrilleSudoku Solvers'	SolverHumain	Hard	5,070.43 ms	1,740.41 ms	95.398 ms	5,010.80 ms	5,180.46 ms	5,020.04 ms	9
'Benchmarking GrilleSudoku Solvers'	Z3Solver	Hard	6,586.97 ms	9,883.60 ms	541.753 ms	6,193.87 ms	7,204.94 ms	6,362.09 ms	10
'Benchmarking GrilleSudoku Solvers'	SolverHumain	Medium	7,256.41 ms	1,901.10 ms	104.206 ms	7,187.02 ms	7,376.23 ms	7,205.97 ms	11

La librairie: Dlxlib

DlxLib 1.3.0

DlxLib is a C# class library that solves exact cover problems by implementing Donald E. Knuth's Algorithm X using the Dancing Links technique.



Quelques parties de notre code

```
namespace Sudoku.DL
{
    0 références
    public class DLSolver : ISudokuSolver
    {

        private static DlxSudokuSolver s = new DlxSudokuSolver();

        8 références
        public void Solve(GrilleSudoku grid)
        {
            s.Solve(grid);
        }

    }
}
```

```
1 référence
class DlxSudokuSolver
{
    1 référence
    public void Solve(GrilleSudoku trav)
    {
        Dlx.MatrixList s = new Dlx.MatrixList(ConvertToMatrix(trav));
        s.search();
        trav.setSudoku(s.convertMatrixSudoku());
    }

    1 référence
    public int[][] ConvertToMatrix(GrilleSudoku grille)
    {
        int[][] sud = new int[9][];
        for (int i = 0; i < 9; i++)
        {
            sud[i] = new int[9];
            for (int j = 0; j < 9; j++)
            {
                sud[i][j] = grille.GetCellule(i, j);
            }
        }
        return sud;
    }
}
```

```
3 références
public class MatrixList
{
    private MatrixNodeHead root;
    //public LinkedList<int> rows = new LinkedList<int>();
    private bool stop = false;
    private LinkedList<MatrixNode> O = new LinkedList<MatrixNode>();
    private int[][] sudoku;

    0 références
    public MatrixList()...

    1 référence
    public MatrixList(int[][] sudoku) //constructeur...

    1 référence
    public int[][] convertMatrixSudoku()...
    2 références
    private int calcRCConstrain(int i, int j)...

    2 références
    private int calcRNConstrain(int i, int value) //contrainte ligne nombre...

    2 références
    private int calcCNConstrain(int j, int value) //contrainte colonne nombre...

    2 références
    private int calcBNConstrain(int i, int j, int value) // contrainte boite nombre...

    2 références
    private void search(int k)  // algorithme de resolution ...

    1 référence
    public void search()...

    2 références
    private void cover(MatrixNodeHead node)...

    2 références
    private void uncover(MatrixNodeHead node)...
}
```

Gestion du projet

03

Etapes du projet

Recherches personnelles sur notre mode de résolution

WEEK 1

WEEK 2

Aide du premier lien git: projet fonctionnel mais non optimisé

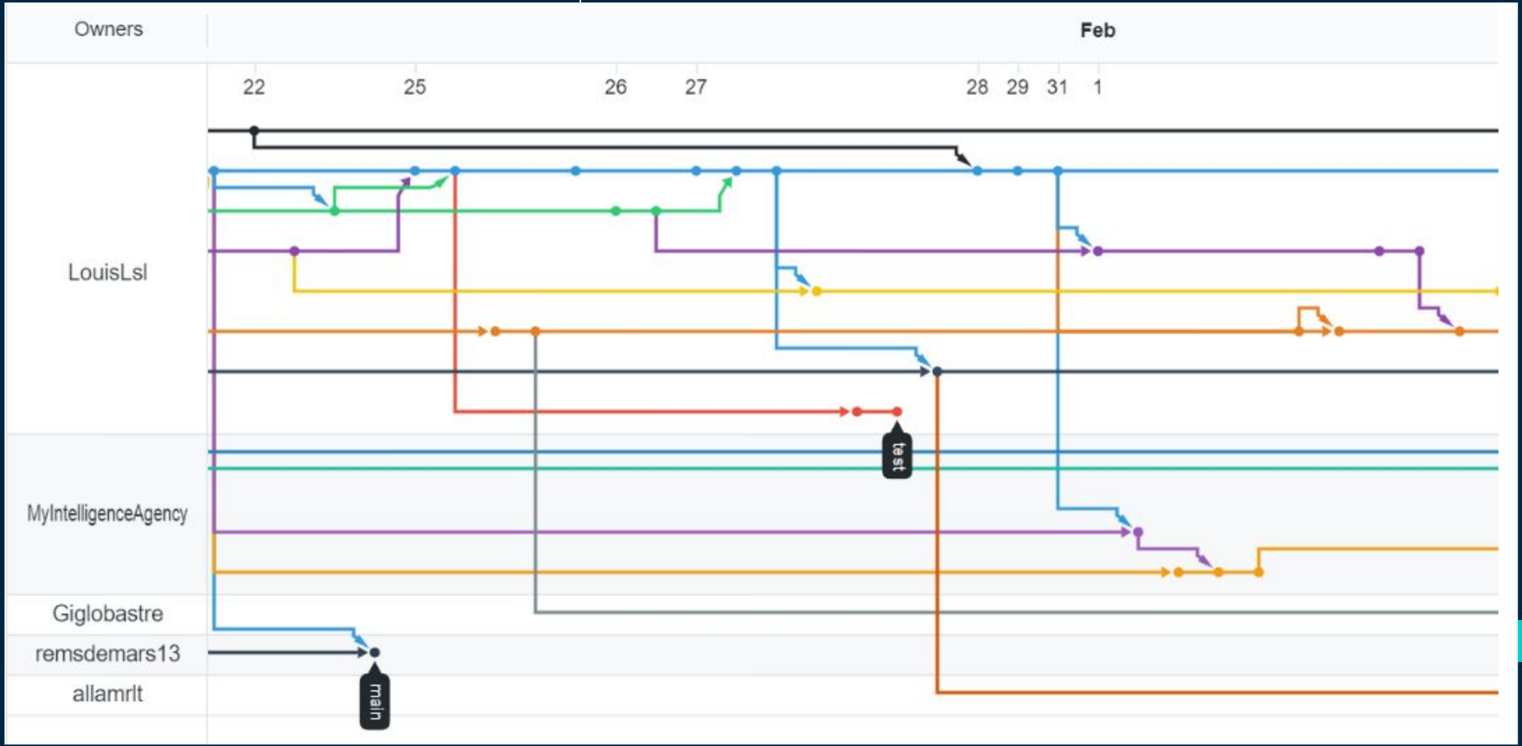
Aide du second lien git: optimisation et reprise de certains bouts de code

WEEK 2

WEEK 3

Dernières mises au point et résultat final

GitHub



Owners
Feb
22
25
26
27
28
29
31
1
LouisLsl
MyIntelligenceAgency
Giglobastre
remsdemars13
allamrlt
test
main

Les résultats (en debug)

- Difficulté facile

```
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: 0,7241 ms
```

- Difficulté moyenne

```
Valid solution:
-------------------------------
| 7  9  6 | 1  5  2 | 3  8  4 |
|         |         |         |
| 5  3  1 | 4  6  8 | 9  2  7 |
|         |         |         |
| 4  2  8 | 3  7  9 | 6  5  1 |
-------------------------------
| 1  5  2 | 6  3  4 | 7  9  8 |
|         |         |         |
| 3  8  4 | 7  9  1 | 2  6  5 |
|         |         |         |
| 9  6  7 | 2  8  5 | 1  4  3 |
-------------------------------
| 2  1  9 | 8  4  3 | 5  7  6 |
|         |         |         |
| 6  4  5 | 9  1  7 | 8  3  2 |
|         |         |         |
| 8  7  3 | 5  2  6 | 4  1  9 |
-------------------------------

Time to solution: 0,6385 ms
```

- Difficulté difficile

```
Valid solution:
-------------------------------
| 4  1  7 | 3  6  9 | 5  2  8 |
|         |         |         |
| 8  3  9 | 1  2  5 | 7  4  6 |
|         |         |         |
| 6  5  2 | 7  4  8 | 3  1  9 |
-------------------------------
| 9  2  5 | 8  3  7 | 4  6  1 |
|         |         |         |
| 7  4  1 | 9  5  6 | 8  3  2 |
|         |         |         |
| 3  8  6 | 4  1  2 | 9  5  7 |
-------------------------------
| 2  9  4 | 6  8  3 | 1  7  5 |
|         |         |         |
| 5  7  3 | 2  9  1 | 6  8  4 |
|         |         |         |
| 1  6  8 | 5  7  4 | 2  9  3 |
-------------------------------

Time to solution: 1,0981 ms
```

Sources

- https://github.com/taylorjg/SudokuDlx
- https://github.com/Bombounet/SudokuIA2.git
- https://fr.wikipedia.org/wiki/Algorithme_X_de_Knuth
- https://lexfridman.com/donald-knuth/
- https://fr.wikipedia.org/wiki/Probl%C3%A8me_de_la_couverture_exacte#Sudoku
- https://github.com/taylorjg/DlxLib
- https://docs.microsoft.com/en-us/dotnet/csharp/
- https://docs.microsoft.com/fr-fr/dotnet/api/system.collections.immutable.immutablelist-1?view=net-5.0