Rapport - Projet Julia

https://github.com/mathusanm6/ProjectJulia

Arborescence

```
ProjectJulia/
├── Manifest.toml           # Versions exactes des dépendances utilisées par le package
├── Project.toml            # Dépendances et métadonnées du projet
├── README.md
├── Rapport.pdf
├── main.jl                 # Point d'entrée de l'application
├── src/                    # Répertoire du code source
│   ├── ProjectJulia.jl     # Fichier du module principal
│   ├── examples.jl         # Implémentations d'exemples
│   ├── generator.jl        # Génération de nouveaux exemples
│   ├── optimizer.jl        # Algorithme d'optimisation
│   ├── types.jl            # Définitions des types
│   └── validator.jl        # Validation des entrées
└── test/                   # Répertoire des tests
    ├── runtests.jl             # Exécuteur principal des tests
    ├── validate_examples.jl    # Tests des exemples
    ├── validate_generator.jl   # Tests du générateur
    ├── validate_optimizer.jl   # Tests de l'optimiseur
    └── validate_types.jl       # Tests des types
```

Installation

1. Téléchargez le ZIP du projet sur GitHub
2. Afin d'initialiser le projet, exécutez dans la racine du projet

```
julia --project=. -e 'using Pkg; Pkg.add(["JuMP", "HiGHS", "Random", "Test",
"JuliaFormatter"])'
```

Lancement du programme

- Pour lancer le programme principal, exécutez

```
julia --project=. main.jl <arg_1> <arg_2> <arg_3> <arg_4>

Usage:
  julia --project=. main.jl examples
  julia --project=. main.jl generate all <size>
  julia --project=. main.jl generate random <size> [seed]
```

- Pour lancer les tests unitaires, exécutez

```
julia --project=. -e "using Pkg; Pkg.test()"
```

Nouvelles structures de données

```
mutable struct Port
    top::Bool
    right::Bool
    bottom::Bool
    left::Bool
end

mutable struct Cell
    entry::Port
    exit::Port
end

mutable struct Circuit
    grid::Array{Cell,2}
end

mutable struct Grid
    size::Int
    circuit::Circuit
    row_constraints::Vector{Int}
    column_constraints::Vector{Int}
end
```

Exemple de définition d'une grille valide

```
grid = Grid(2, Circuit(2), [2, 2], [2, 2])

grid.circuit.grid[1, 1] =
    Cell(Port(false, false, true, false), Port(false, true, false, false))
grid.circuit.grid[1, 2] =
    Cell(Port(false, false, false, true), Port(false, false, true, false))

grid.circuit.grid[2, 1] =
    Cell(Port(false, true, false, false), Port(true, false, false, false))
grid.circuit.grid[2, 2] =
    Cell(Port(true, false, false, false), Port(false, false, false, true))

# == Affichage == #
       2    2
     +---++---+
     |   ||   |
  2  | **||** |
     | * || * |
     +---++---+
     | * || * |
  2  | **||** |
     |   ||   |
     +---++---+
```

Variables de décision

- entry_top[i, j] : Variable binaire indiquant si la cellule à la position (i, j) possède une entrée par le haut.
- entry_right[i, j] : Variable binaire indiquant si la cellule à la position (i, j) possède une entrée par la droite.
- $\text{entry\_bottom[i, j]}$: Variable binaire indiquant si la cellule à la position (i, j) possède une entrée par le bas.
- entry_left[i, j] : Variable binaire indiquant si la cellule à la position (i, j) possède une entrée par la gauche.
- exit_top[i, j] : Variable binaire indiquant si la cellule à la position (i, j) possède une sortie par le haut.
- exit_right[i, j] : Variable binaire indiquant si la cellule à la position (i, j) possède une sortie par la droite.
- exit_bottom[i, j] : Variable binaire indiquant si la cellule à la position (i, j) possède une sortie par le bas.
- exit_left[i, j] : Variable binaire indiquant si la cellule à la position (i, j) possède une sortie par la gauche.

Soit n la taille de la grille. Il y a n^2 cases avec chacune 8 variables. Au total, il y aurait $8 \cdot n^2$ variables.

Contraintes

1. **Contraintes de lignes :**
   - Chaque ligne doit contenir exactement le nombre spécifié de cellules non-vides (selon les contraintes de ligne).

     Pour chaque ligne $i \in \{1, ..., n\}$,
     $\sum_{j=1}^{n} (\text{entry\_top[i, j]} + \text{entry\_right[i, j]} + \text{entry\_bottom[i, j]} + \text{entry\_left[i, j]}) = R_i$

2. **Contraintes de colonnes :**
   - Chaque colonne doit contenir exactement le nombre spécifié de cellules non-vides (selon les contraintes de colonne).

     Pour chaque colonne $j \in \{1, ..., n\}$,
     $\sum_{i=1}^{n} (\text{entry\_top[i, j]} + \text{entry\_right[i, j]} + \text{entry\_bottom[i, j]} + \text{entry\_left[i, j]}) = R_j$

3. **Contraintes par cellule :**
   - Une cellule peut avoir au maximum une direction d'entrée.

     Pour chaque cellule $(i, j) \in \{1, ..., n\}^2$,
     $\text{entry\_top[i, j]} + \text{entry\_right[i, j]} + \text{entry\_bottom[i, j]} + \text{entry\_left[i, j]} \leq 1$
   - Une cellule peut avoir au maximum une direction de sortie.

     Pour chaque cellule $(i, j) \in \{1, ..., n\}^2$,

$$\text{exit\_top}[i,\ j] + \text{exit\_right}[i,\ j] + \text{exit\_bottom}[i,\ j] + \text{exit\_left}[i,\ j] \leq 1$$

- Une cellule ne peut pas avoir une entrée et une sortie dans la même direction.

Pour chaque cellule $(i, j) \in \{1, ..., n\}^2$,

$$\text{entry\_top}[i,\ j] + \text{exit\_top}[i,\ j] \leq 1$$
$$\text{entry\_right}[i,\ j] + \text{exit\_ right}[i,\ j] \leq 1$$
$$\text{entry\_bottom}[i,\ j] + \text{exit\_bottom}[i,\ j] \leq 1$$
$$\text{entry\_left}[i,\ j] + \text{exit\_left}[i,\ j] \leq 1$$

- Une cellule doit avoir un nombre égal de directions d'entrée et de sortie (0 ou 1).

Pour chaque cellule $(i, j) \in \{1, ..., n\}^2$,

$$\text{entry\_top}[i,\ j] + \text{entry\_right}[i,\ j] + \text{entry\_bottom}[i,\ j] + \text{entry\_left}[i,\ j] =$$
$$\text{exit\_top}[i,\ j] + \text{exit\_right}[i,\ j] + \text{exit\_bottom}[i,\ j] + \text{exit\_left}[i,\ j]$$

4. **Contraintes de connectivité :**
   - Une sortie vers le haut de la cellule (i, j) doit correspondre à une entrée par le bas de la cellule $(i - 1, j)$.

Si $i > 1$:

$$\text{exit\_top}[i,\ j] = \text{entry\_bottom}[i\ -\ 1,\ j]$$

Sinon:

$$\text{exit\_top}[i,\ j] = 0$$

   - Une sortie vers la droite de la cellule (i, j) doit correspondre à une entrée par la gauche de la cellule $(i, j + 1)$.

Si $j < n$:

$$\text{exit\_right}[i,\ j] = \text{entry\_left}[i,\ j\ +\ 1]$$

Sinon:

$$\text{exit\_right}[i,\ j] = 0$$

   - Une sortie vers le bas de la cellule (i, j) doit correspondre à une entrée par le haut de la cellule $(i + 1, j)$.

Si $i < n$:

$$\text{exit\_bottom}[i,\ j] = \text{entry\_top}[i\ +\ 1,\ j]$$

Sinon:

$$\text{exit\_bottom}[i,\ j] = 0$$

   - Une sortie vers la gauche de la cellule (i, j) doit correspondre à une entrée par la droite de la cellule $(i, j - 1)$.

Si $j > 1$:

$$\text{exit\_left}[i,\ j] = \text{entry\_right}[i,\ j\ -\ 1]$$

Sinon:

$$\text{exit\_left}[i,\ j] = 0$$

- Les cellules au bord de la grille ne peuvent pas avoir de sorties vers l'extérieur.

5. **Contraintes pour éliminer les sous-tours :**
   - Des contraintes supplémentaires sont ajoutées pour éliminer les sous-tours détectés. La détection des sous-tours se fait après chaque optimisation. On en choisit une à interdire avec la contrainte sur la longueur (similaire à l'exemple du TSP des travaux pratiques).

Pour chaque boucle composée de k cellules $(i_1, j_1), ..., (i_k, j_k)$:

$$\sum_{\ell=1}^{k}(\text{entry\_top}[i_\ell,\ j_\ell] + \text{entry\_right}[i_\ell,\ j_\ell] + \text{entry\_bottom}[i_\ell,\ j_\ell] + \text{entry\_left}[i_\ell,\ j_\ell] + \text{exit\_top}[i_\ell,\ j_\ell] + \text{exit\_right}[i_\ell,\ j_\ell] + \text{exit\_bottom}[i_\ell,\ j_\ell] + \text{exit\_left}[i_\ell,\ j_\ell]) \leq 2k - 1$$

Fonction objective

La fonction objective n'est pas importante car il s'agit principalement d'un problème de faisabilité plutôt que d'optimisation.

Méthode efficace pour valider les solutions proposées par l'optimiseur

Afin d'automatiser la validation des solutions obtenues à la fin d'exécution de l'optimiseur, j'ai défini des fonctions spécialisées dans le fichier `validator.jl` . La plus importante de ces fonctions est `check_valid_grid` qui est utilisée à plusieurs reprises lors des tests unitaires.

Un choix critique

Dans un premier temps, après avoir determiné les sous-tours, j'ai relancé l'optimisation avec de nouvelles contraintes interdisant tous ces sous-tours, mais aucune solution réalisable n'a pu être identifiée. Pour y remédier, j'ai opté pour un ajout progressif de contraintes, avec une nouvelle optimisation à chaque étape. La solution est atteinte lorsqu'il ne reste plus qu'un seul tour.

Solutions

Example (Vérifié)

```
Solution :
       5    5    2    2    4
     +---++---++---++---++---+
     |   ||   ||   ||   ||   |
  5  | **||***||***||***||** |
     | * ||   ||   ||   || * |
     +---++---++---++---++---+
     | * ||   ||   ||   || * |
  3  | **||** ||   ||   || * |
     |   || * ||   ||   || * |
     +---++---++---++---++---+
     |   || * ||   ||   || * |
  3  | **||** ||   ||   || * |
     | * ||   ||   ||   || * |
     +---++---++---++---++---+
     | * ||   ||   ||   || * |
  5  | * || **||***||***||** |
     | * || * ||   ||   ||   |
     +---++---++---++---++---+
     | * || * ||   ||   ||   |
  2  | **||** ||   ||   ||   |
     |   ||   ||   ||   ||   |
     +---++---++---++---++---+
```

Jeu 1

```
Solution :
       3    4    3    3    5
     +---++---++---++---++---+
     |   ||   ||   ||   ||   |
  4  | **||** ||   || **||** |
     | * || * ||   || * || * |
     +---++---++---++---++---+
     | * || * ||   || * || * |
  5  | * || **||***||** || * |
     | * ||   ||   ||   || * |
     +---++---++---++---++---+
     | * ||   ||   ||   || * |
  3  | **||** ||   ||   || * |
     |   || * ||   ||   || * |
     +---++---++---++---++---+
     |   || * ||   ||   || * |
  3  |   || **||** ||   || * |
     |   ||   || * ||   || * |
     +---++---++---++---++---+
     |   ||   || * ||   || * |
  3  |   ||   || **||***||** |
     |   ||   ||   ||   ||   |
```

```
     +---++---++---++---++---+
```

Jeu 2

```
Solution :
       3    2    4    5    4
     +---++---++---++---++---+
     |   ||   ||   ||   ||   |
  3  |   ||   || **||***||** |
     |   ||   || * ||   || * |
     +---++---++---++---++---+
     |   ||   || * ||   || * |
  5  | **||***||** || **||** |
     | * ||   ||   || * ||   |
     +---++---++---++---++---+
     | * ||   ||   || * ||   |
  2  | * ||   ||   || * ||   |
     | * ||   ||   || * ||   |
     +---++---++---++---++---+
     | * ||   ||   || * ||   |
  5  | **||***||** || **||** |
     |   ||   || * ||   || * |
     +---++---++---++---++---+
     |   ||   || * ||   || * |
  3  |   ||   || **||***||** |
     |   ||   ||   ||   ||   |
     +---++---++---++---++---+
```

Jeu 3

```
Solution :
       4    4    3    3    4
     +---++---++---++---++---+
     |   ||   ||   ||   ||   |
  3  |   ||   || **||***||** |
     |   ||   || * ||   || * |
     +---++---++---++---++---+
     |   ||   || * ||   || * |
  4  | **||***||** ||   || * |
     | * ||   ||   ||   || * |
     +---++---++---++---++---+
     | * ||   ||   ||   || * |
  5  | * || **||***||** || * |
     | * || * ||   || * || * |
     +---++---++---++---++---+
     | * || * ||   || * || * |
  4  | * || * ||   || **||** |
     | * || * ||   ||   ||   |
     +---++---++---++---++---+
     | * || * ||   ||   ||   |
  2  | **||** ||   ||   ||   |
```

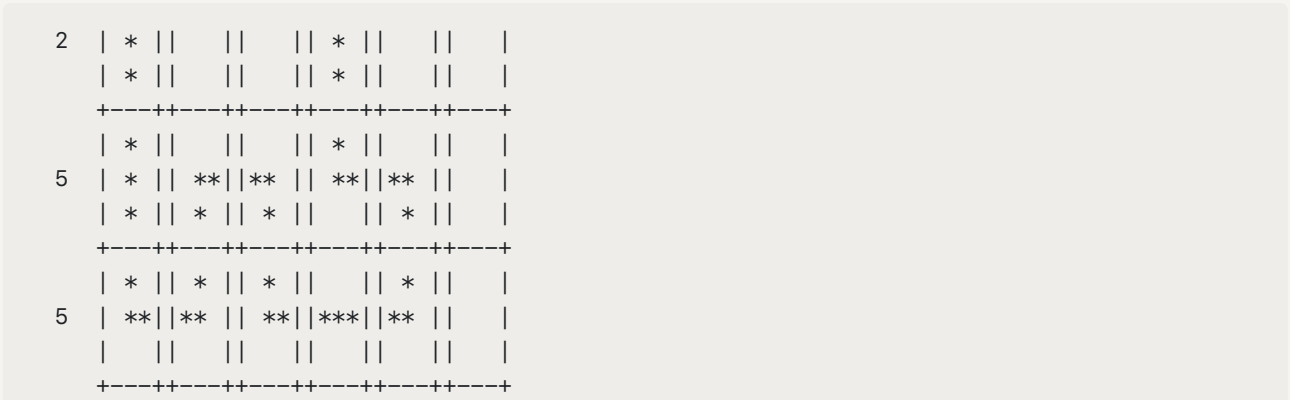
```
     |   ||   ||   ||   ||   |
     +---++---++---++---++---+
```

Jeu 4

```
Solution :
       4    4    2    3    2    5
     +---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   |
  3  |   ||   ||   || **||***||** |
     |   ||   ||   || * ||   || * |
     +---++---++---++---++---++---+
     |   ||   ||   || * ||   || * |
  4  |   || **||***||** ||   || * |
     |   || * ||   ||   ||   || * |
     +---++---++---++---++---++---+
     |   || * ||   ||   ||   || * |
  3  | **||** ||   ||   ||   || * |
     | * ||   ||   ||   ||   || * |
     +---++---++---++---++---++---+
     | * ||   ||   ||   ||   || * |
  2  | * ||   ||   ||   ||   || * |
     | * ||   ||   ||   ||   || * |
     +---++---++---++---++---++---+
     | * ||   ||   ||   ||   || * |
  6  | * || **||***||***||***||** |
     | * || * ||   ||   ||   ||   |
     +---++---++---++---++---++---+
     | * || * ||   ||   ||   ||   |
  2  | **||** ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---+
```

Jeu 5

```
Solution :
       4    5    4    5    5    3
     +---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   |
  4  |   || **||** ||   || **||** |
     |   || * || * ||   || * || * |
     +---++---++---++---++---++---+
     |   || * || * ||   || * || * |
  5  |   || * || **||***||** || * |
     |   || * ||   ||   ||   || * |
     +---++---++---++---++---++---+
     |   || * ||   ||   ||   || * |
  5  | **||** ||   || **||***||** |
     | * ||   ||   || * ||   ||   |
     +---++---++---++---++---++---+
     | * ||   ||   || * ||   ||   |
```


2
5
5

Génération de toutes les grilles valides d'une certaine taille

Méthode choisie

Faire varier les contraintes sur les lignes et sur les colonnes dans un ordre lexicographique pour voir s'il est possible de trouver une solution. Si c'est le cas, alors on a trouvé une possibilité.

J'ai opté pour générer toutes les possibilités avec une taille de grille donnée. Dans ce cadre, n = 3 est la dernière à se terminer en un temps court (< 1 min). Dès n = 4, le nombre de contraintes possibles explosent ($5^{4^2} = 5^16 = 1,526 \times 10^{11}$ si on considère toutes les combinaisons des contraintes pour les lignes et les colonnes).

Il s'agit donc d'une méthode qui, en l'état actuel, manque d'efficacité. Il serait pertinent d'y intégrer une forme d'heuristique, en identifiant certaines combinaisons comme contradictoires au regard des règles du problème. Par exemple, si toutes les contraintes associées aux lignes sont nulles, alors il devient impossible d'obtenir une valeur entière non nulle dans les contraintes des colonnes.

Génération de toutes les solutions pour n = 4

Il a fallu environ 9 min pour que n = 4 se termine. Voici trois parmi tant d'autres de ces grilles:

```
Original Grid:
Solution :
       4    3    3    4
     +---++---++---++---+
     |   ||   ||   ||   |
  4  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+
     |   ||   ||   ||   |
  2  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+
     |   ||   ||   ||   |
  4  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+
     |   ||   ||   ||   |
  4  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+

Solved Grid:
Solution :
       4    3    3    4
     +---++---++---++---+
     |   ||   ||   ||   |
  4  | **||***||***||** |
     | * ||   ||   || * |
     +---++---++---++---+
```

```
     | * ||   ||   || * |
  2  | * ||   ||   || * |
     | * ||   ||   || * |
     +---++---++---++---+
     | * ||   ||   || * |
  4  | * || **||** || * |
     | * || * || * || * |
     +---++---++---++---+
     | * || * || * || * |
  4  | **||** || **||** |
     |   ||   ||   ||   |
     +---++---++---++---+

Is the solved grid valid? true

Original Grid:
Solution :
       4    4    4    2
     +---++---++---++---+
     |   ||   ||   ||   |
  3  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+
     |   ||   ||   ||   |
  3  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+
     |   ||   ||   ||   |
  4  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+
     |   ||   ||   ||   |
  4  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+

Solved Grid:
Solution :
       4    4    4    2
     +---++---++---++---+
     |   ||   ||   ||   |
  3  | **||***||** ||   |
     | * ||   || * ||   |
     +---++---++---++---+
     | * ||   || * ||   |
  3  | **||** || * ||   |
     |   || * || * ||   |
     +---++---++---++---+
     |   || * || * ||   |
  4  | **||** || **||** |
     | * ||   ||   || * |
     +---++---++---++---+
     | * ||   ||   || * |
```

```
  4  | **||***||***||** |
     |   ||   ||   ||   |
     +---++---++---++---+

Is the solved grid valid? true

Original Grid:
Solution :
       4    3    4    3
     +---++---++---++---+
     |   ||   ||   ||   |
  3  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+
     |   ||   ||   ||   |
  3  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+
     |   ||   ||   ||   |
  4  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+
     |   ||   ||   ||   |
  4  |   ||   ||   ||   |
     |   ||   ||   ||   |
     +---++---++---++---+

Solved Grid:
Solution :
       4    3    4    3
     +---++---++---++---+
     |   ||   ||   ||   |
  3  | **||***||** ||   |
     | * ||   || * ||   |
     +---++---++---++---+
     | * ||   || * ||   |
  3  | * ||   || **||** |
     | * ||   ||   || * |
     +---++---++---++---+
     | * ||   ||   || * |
  4  | * || **||** || * |
     | * || * || * || * |
     +---++---++---++---+
     | * || * || * || * |
  4  | **||** || **||** |
     |   ||   ||   ||   |
     +---++---++---++---+

Is the solved grid valid? true
```

Génération aléatoire de grille valide pour une taille donnée

Dans ce cas-ci, nous générons des grilles de taille donnée de manière aléatoire jusqu'à ce que la grille générée soit solvable. Si c'est le cas, on retourne la grille.

Une optimisation simple était de voir si la combinaison des contraintes était déjà traitée pour éviter des calculs non nécessaires.

```
julia --project=. main.jl generate random 6
Generating Random Solvable Grid...
Original Grid:
Solution :
       5    3    3    5    6    4
     +---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   |
  6  |   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   |
  5  |   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   |
  2  |   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   |
  6  |   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   |
  5  |   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   |
  2  |   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---+

Solved Grid:
Solution :
       5    3    3    5    6    4
     +---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   |
  6  | **||***||** || **||***||** |
     | * ||   || * || * ||   || * |
     +---++---++---++---++---++---+
     | * ||   || * || * ||   || * |
  5  | * ||   || **||** || **||** |
     | * ||   ||   ||   || * ||   |
     +---++---++---++---++---++---+
     | * ||   ||   ||   || * ||   |
  2  | * ||   ||   ||   || * ||   |
```

```
     | * ||   ||   ||   || * ||   |
     +---++---++---++---++---++---+
     | * ||   ||   ||   || * ||   |
  6  | * || **||***||** || **||** |
     | * || * ||   || * ||   || * |
     +---++---++---++---++---++---+
     | * || * ||   || * ||   || * |
  5  | **||** ||   || * || **||** |
     |   ||   ||   || * || * ||   |
     +---++---++---++---++---++---+
     |   ||   ||   || * || * ||   |
  2  |   ||   ||   || **||** ||   |
     |   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---+

Is the solved grid valid? true
========================
```

Ce calcul a duré pendant 1m18s.

```
julia --project=. main.jl generate random 7
Generating Random Solvable Grid...
Original Grid:
Solution :
       6    5    7    6    6    6    2
     +---++---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   ||   |
  7  |   ||   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   ||   |
  6  |   ||   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   ||   |
  3  |   ||   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   ||   |
  6  |   ||   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   ||   |
  6  |   ||   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   ||   |
  6  |   ||   ||   ||   ||   ||   ||   |
     |   ||   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   ||   |
  4  |   ||   ||   ||   ||   ||   ||   |
```

```
     |   ||   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---++---+

Solved Grid:
Solution :
       6    5    7    6    6    6    2
     +---++---++---++---++---++---++---+
     |   ||   ||   ||   ||   ||   ||   |
  7  | **||***||***||***||***||***||** |
     | * ||   ||   ||   ||   ||   || * |
     +---++---++---++---++---++---++---+
     | * ||   ||   ||   ||   ||   || * |
  6  | * ||   || **||***||***||***||** |
     | * ||   || * ||   ||   ||   ||   |
     +---++---++---++---++---++---++---+
     | * ||   || * ||   ||   ||   ||   |
  3  | * ||   || **||** ||   ||   ||   |
     | * ||   ||   || * ||   ||   ||   |
     +---++---++---++---++---++---++---+
     | * ||   ||   || * ||   ||   ||   |
  6  | **||** || **||** || **||** ||   |
     |   || * || * ||   || * || * ||   |
     +---++---++---++---++---++---++---+
     |   || * || * ||   || * || * ||   |
  6  | **||** || **||***||** || * ||   |
     | * ||   ||   ||   ||   || * ||   |
     +---++---++---++---++---++---++---+
     | * ||   ||   ||   ||   || * ||   |
  6  | **||** || **||***||** || * ||   |
     |   || * || * ||   || * || * ||   |
     +---++---++---++---++---++---++---+
     |   || * || * ||   || * || * ||   |
  4  |   || **||** ||   || **||** ||   |
     |   ||   ||   ||   ||   ||   ||   |
     +---++---++---++---++---++---++---+

Is the solved grid valid? true
=========================
```

Ce calcul a duré pendant 8m5s.