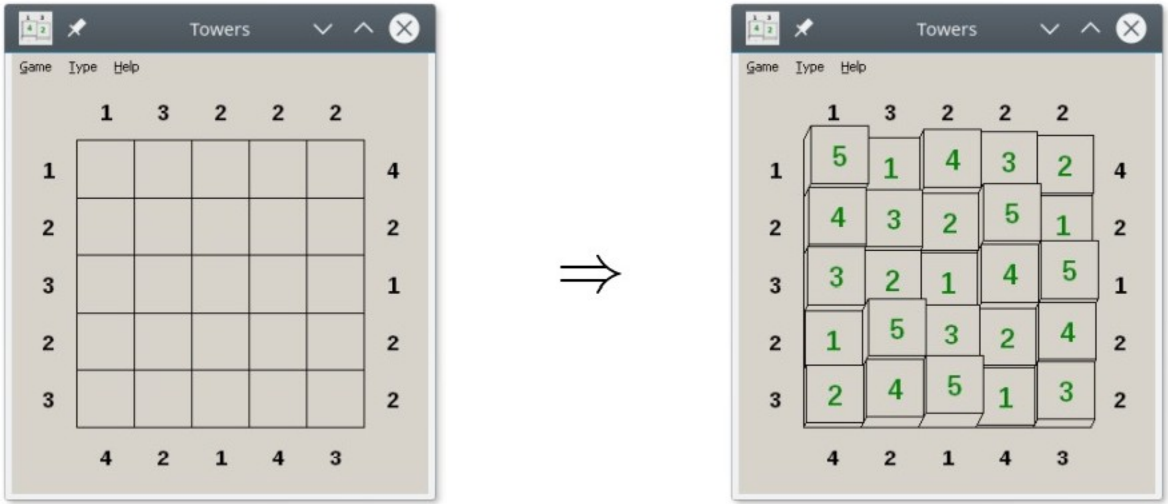
Towers

Towers

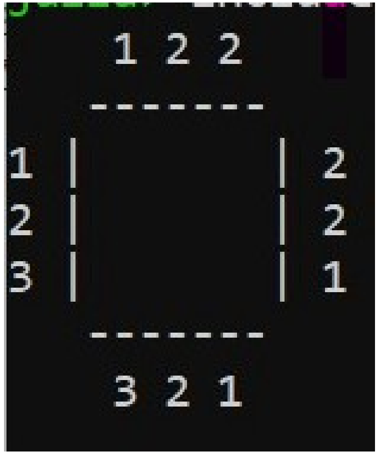


Exemple de grille initiale

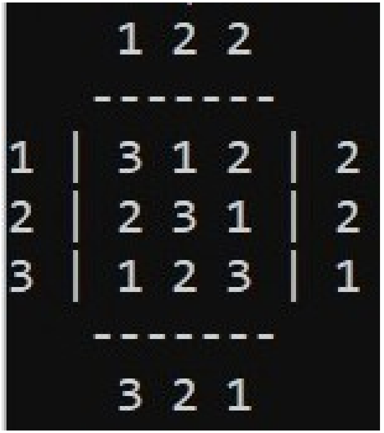
Exemple de grille résolue

Modélisation

- Grille : Matrice n*n
- nord, sud, est, ouest : vecteurs n



Nord=[1 2 2]
Sud=[3 2 1]
Est=[2 2 1]
Ouest=[1 2 3]



Générer une grille

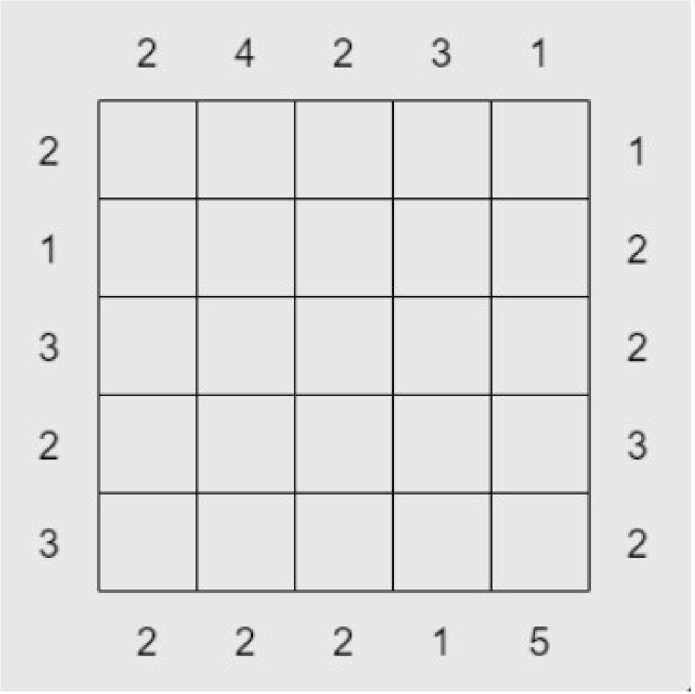
- On rempli une grille n*n de sorte qu’il n’y ait aucun doublons sur les lignes et sur les colonnes

1	3	2	4
3	2		

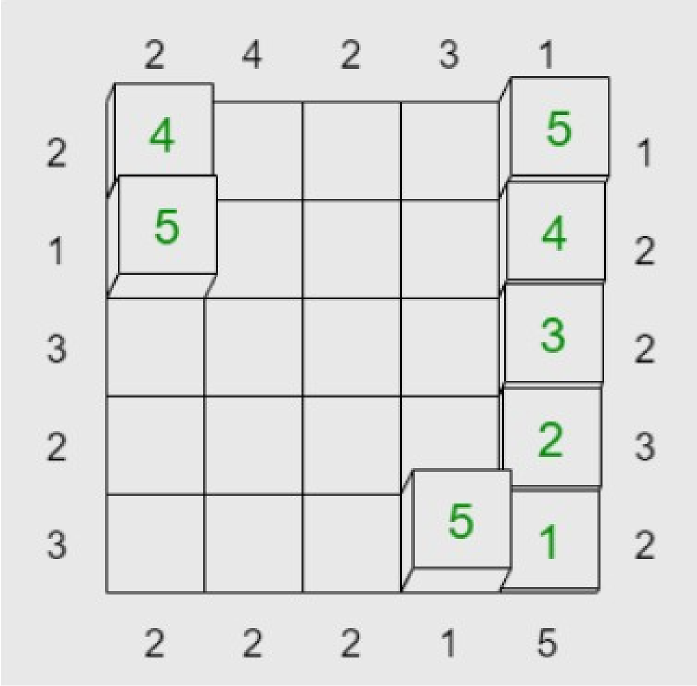
Ici on choisit aléatoirement entre 1 et 4

- On calcule les tours visibles depuis chaque directions, qu’on stocke dans les vecteurs nord, est sud et ouest

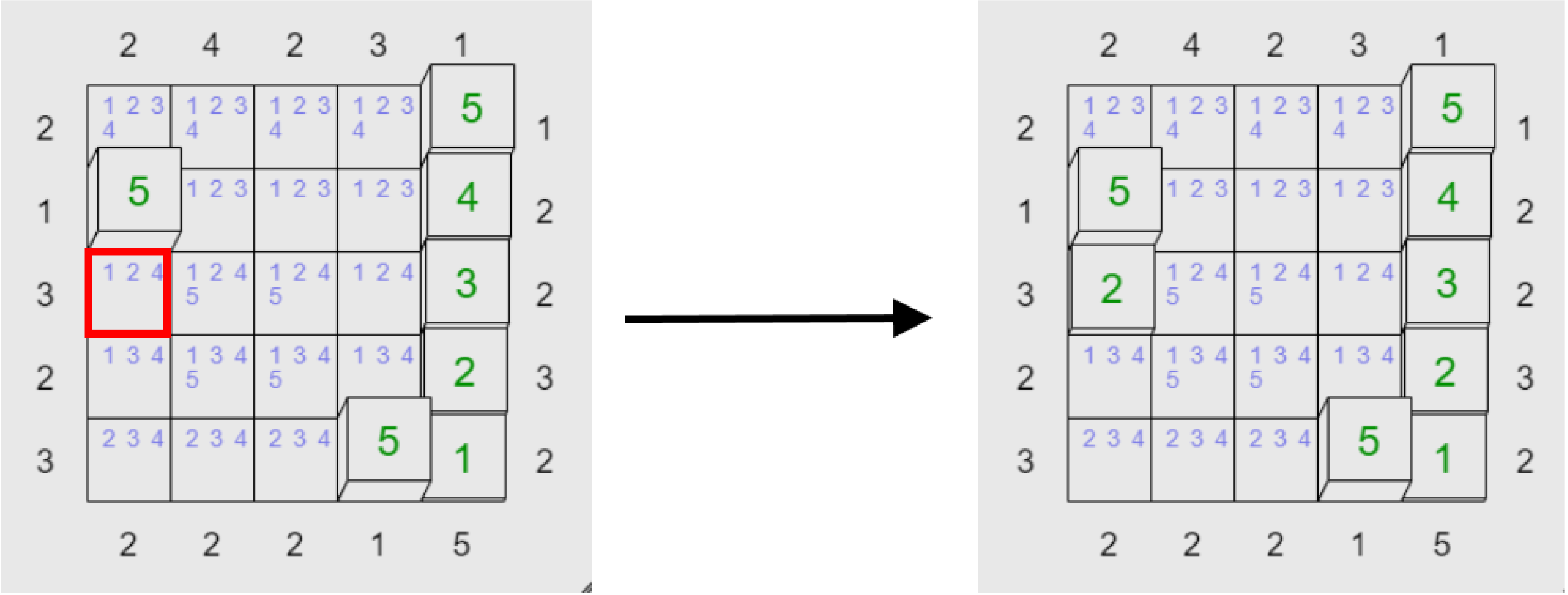
Heuristique



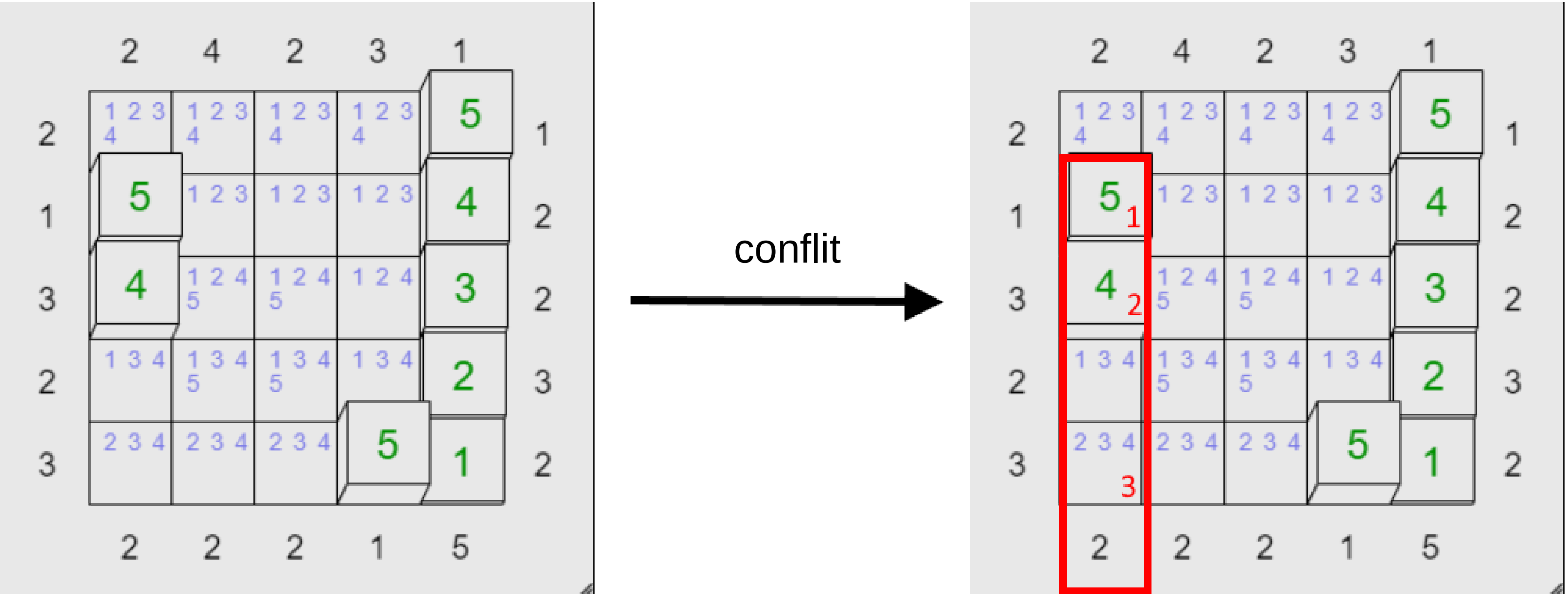
Initialisation



Heuristique



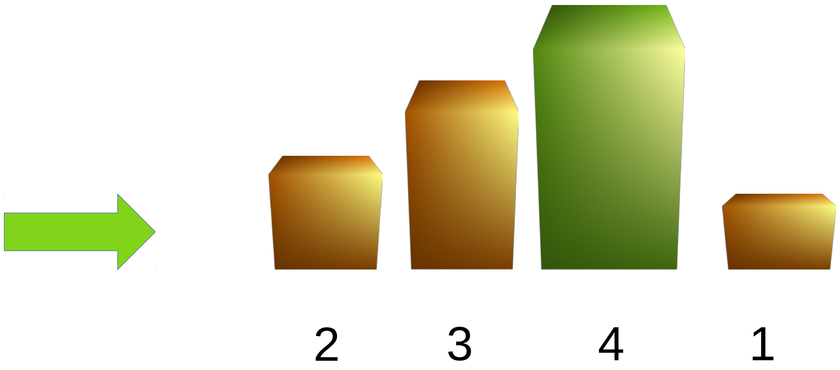
Heuristique



Cplex

```
@variable(m,x[1:n,1:n,1:n],Bin) # ==1 ssi (i,j) contient k
@variable(m,yn[1:n,1:n],Bin)    # ==1 ssi (i,j) visible depuis le nord
@variable(m,ys[1:n,1:n],Bin)    # ==1 ssi (i,j) visible depuis le sud
@variable(m,yo[1:n,1:n],Bin)    # ==1 ssi (i,j) visible depuis l'ouest
@variable(m,ye[1:n,1:n],Bin)    # ==1 ssi (i,j) visible depuis l'est
```

```
#Une seule valeur par case
@constraint(m, [i in 1:n, j in 1:n], sum(x[i,j,k] for k in 1:n) == 1)
#Chiffres différents sur une ligne
@constraint(m, [i in 1:n, k in 1:n], sum(x[i,j,k] for j in 1:n) == 1)
#Chiffres différents sur une colonne
@constraint(m, [j in 1:n, k in 1:n], sum(x[i,j,k] for i in 1:n) == 1)
```



La tour en (i,j) est visible depuis l’ouest ssi

$$\sum_{j_- < j} \sum_{k_- > k} x[i, j, k] = 0$$

on a toujours

$$0 \leq \frac{\sum_{j\_<j} \sum_{k\_>k} x[i,j,k]}{2n} < 1$$

$$1 - \frac{\sum_{j\_<j} \sum_{k\_>k} x[i,j,k]}{2n} = 1 \text{ ssi } \frac{\sum_{j\_<j} \sum_{k\_>k} x[i,j,k]}{2n} = 0 \text{ ie ssi la tour est visible}$$

$$\text{sinon } 0 \leq 1 - \frac{\sum_{j\_<j} \sum_{k\_>k} x[i,j,k]}{2n} < 1$$

Ainsi la contrainte

```
@constraint(m, [i in 1:n, j in 1:n, k in 1:n], yo[i,j]<=1-sum(x[i,j_,k_] for j_ in 1:j-1 for k_ in k:n)/(2*n)+1-x[i,j,k])
```

$$\text{impose } y[i,j] = 0 \text{ si } \sum_{j\_<j} \sum_{k\_>k} x[i,j,k] > 0$$

1-x[i,j,k] est là pour ne pas imposer y[i,j] pour les k tels que x[i,j,k]==0

De meme

```
@constraint(m, [i in 1:n ,j in 1:n, k in 1:n], yo[i,j]>=1-sum(x[i,j_,k_] for j_ in 1:j-1 for k_ in k:n)-2*n*(1-x[i,j,k]))
```

impose $y[i,j] = 1$ si $\sum_{j\_<j} \sum_{k\_>k} x[i,j,k] = 0$

on a donc bien

$$y[i,j] = 1$$

ssi

$$\sum_{j\_<j} \sum_{k\_>k} x[i,j,k] = 0$$

Finalement

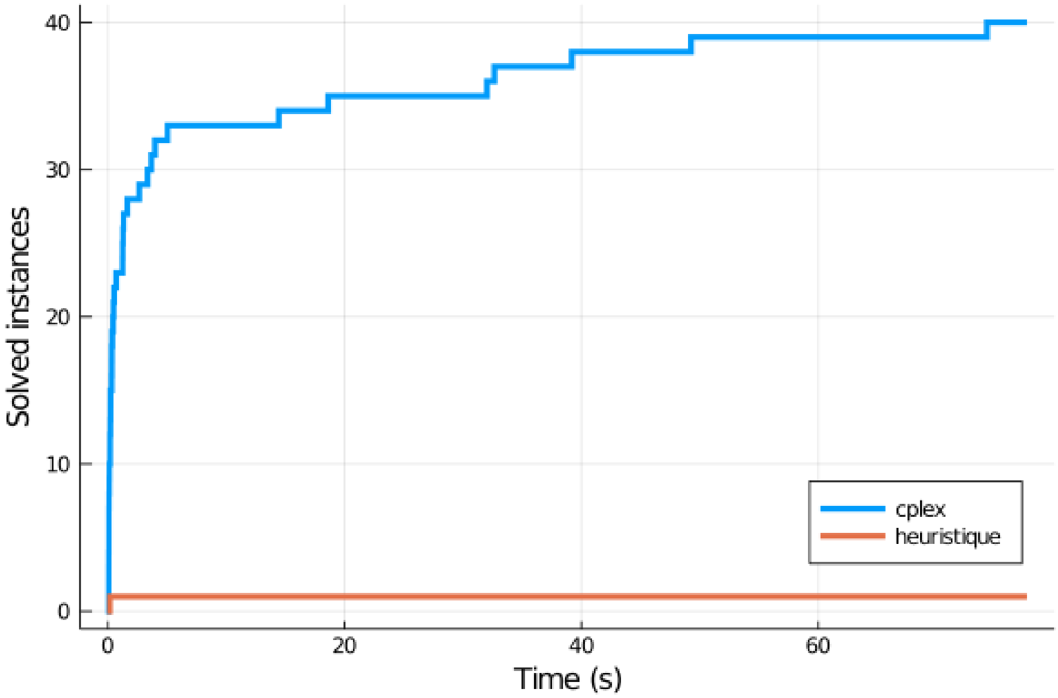
```
#Nord
@constraint(m, [j in 1:n], sum(yn[i,j] for i in 1:n)==nord[j])
@constraint(m, [i in 1:n, j in 1:n, k in 1:n], yn[i,j]<=1-sum(x[i_,j,k_] for i_ in 1:i-1 for k_ in k:n)/(2*n)+1-x[i,j,k])
@constraint(m, [i in 1:n ,j in 1:n, k in 1:n], yn[i,j]>=1-sum(x[i_,j,k_] for i_ in 1:i-1 for k_ in k:n)-2*n*(1-x[i,j,k]))

#Sud
@constraint(m, [j in 1:n], sum(ys[i,j] for i in 1:n)==sud[j])
@constraint(m, [i in 1:n, j in 1:n, k in 1:n], ys[i,j]<=1-sum(x[i_,j,k_] for i_ in i+1:n for k_ in k:n)/(2*n)+1-x[i,j,k])
@constraint(m, [i in 1:n ,j in 1:n, k in 1:n], ys[i,j]>=1-sum(x[i_,j,k_] for i_ in i+1:n for k_ in k:n)-2*n*(1-x[i,j,k]))

#Est
@constraint(m, [i in 1:n], sum(ye[i,j] for j in 1:n)==est[i])
@constraint(m, [i in 1:n, j in 1:n, k in 1:n], ye[i,j]<=1-sum(x[i,j_,k_] for j_ in j+1:n for k_ in k:n)/(2*n)+1-x[i,j,k])
@constraint(m, [i in 1:n ,j in 1:n, k in 1:n], ye[i,j]>=1-sum(x[i,j_,k_] for j_ in j+1:n for k_ in k:n)-2*n*(1-x[i,j,k]))


#Ouest
@constraint(m, [i in 1:n], sum(yo[i,j] for j in 1:n)==ouest[i])
@constraint(m, [i in 1:n, j in 1:n, k in 1:n], yo[i,j]<=1-sum(x[i,j_,k_] for j_ in 1:j-1 for k_ in k:n)/(2*n)+1-x[i,j,k])
@constraint(m, [i in 1:n ,j in 1:n, k in 1:n], yo[i,j]>=1-sum(x[i,j_,k_] for j_ in 1:j-1 for k_ in k:n)-2*n*(1-x[i,j,k]))
```

Résultats



On a généré des instances de taille 5,6,7,8. Elles sont toutes résolues par cplex en environ 1s sauf pour les instance de taille 8 où il faut quelques dizaines de secondes.

L'heuristique quant à elle ne parvient qu'à résoudre une instance de taille 3, au delà cela devient très peu probable.

Singles

3 Singles

Exemple de grille initiale

$\Rightarrow$

Exemple de grille résolue

modélisation

- Grille : Matrice n*n
- Pour le cplex :Y : matrice n*n qui contient zero en i,j si la case i,j de la grille est masquée, 1 sinon

Ex :

```
 -------
| 2 3 2 |
| 2 1 1 |
| 3 2 1 |
 -------
```

Grille=[[2,3,2] [2,1,1] [3,2,1]]

```
 -------
| * 3 2 |
| 2 1 * |
| 3 2 1 |
 -------
```

Grille=[[0,3,2] [2,1,0] [3,2,1]]

Y=[[0,1,1] [1,1,0] [1,1,1]]

Générer une grille

- On place un maximum de cases noires tel que la grille reste connexe (parcours en profondeur d’un graphe)
- On rempli les chiffres des cases blanches tel qu’il n’y ait aucun doublons sur les lignes et sur les colonnes
- On rempli au hasard les cases noires

Heuristique

- On liste les doublons présents dans les lignes et les colonnes

```
-------
| 2 3 2 |
| 2 1 1 |
| 3 2 1 |
-------
```

Doublons=[[(1,1),(1,3)] [(2,2),(2,3)] [(1,1),(2,1)] [(2,3),(3,3)]]

- On noircit dans ces listes de doublons autant de cases que nécessaire pour qu’il n’y ait plus de doublons
- Tant que c’est connexe, on garde, sinon on recommence

```
-------
| 2 3 2 |
| 2 * 1 |
| 3 2 * |
-------
```

```
-------
| 2 3 * |
| * 1 1 |
| 3 2 * |
-------
```

```
-------
| * 3 2 |
| 2 1 * |
| 3 2 1 |
-------
```

Cplex

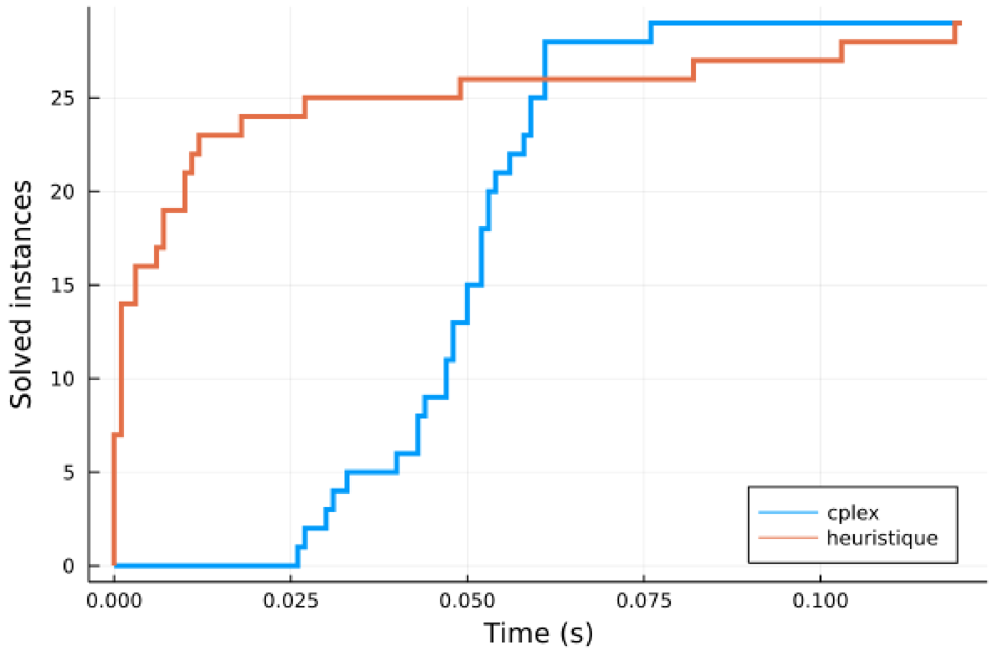
```
@variable(singles,y[1:n,1:n],Bin)   # == 1 ssi (i,j) blanche
@objective(singles,Max,y[1,1])

#Chiffres différents sur une ligne, zero mis à part
@constraint(singles, lin[k in 1:n, i in 1:n], sum(y[i,j] for j in 1:n if x[i,j] == k) <= 1 )
#Chiffres différents sur une colonne, zero mis à part
@constraint(singles, col[k in 1:n, j in 1:n], sum(y[i,j] for i in 1:n if x[i,j]==k) <= 1)

#pas deux cases voisines noires
@constraint(singles, black[i in 1:n-1, j in 1:n], y[i,j]+y[i+1,j] >= 1)
@constraint(singles, black_[j in 1:n-1, i in 1:n], y[i,j]+y[i,j+1] >= 1)


 y_m = JuMP.value.(y)

 #contraintes de connexité
 @constraint(singles, connexite, sum( y[i,j] for i in 1:n for j in 1:n if y_m[i,j] == 0 ) >= 1)
```



n=5,6,7
Dix instances chaques

Au dessus de n=10, heuristique ne trouve pas toujours de solution
Au dessus de n=40 génération prend un peu de temps
Pour n=100, **cplex : 33s**
generation : 5min 43s
Cplex est finalement le moins limitant.