



Projet R0203



Yael GOSSEC - Thomas VERRECCHIA



Jeu 1 : Singles

1	2	1	5	5
1	5	3	1	4
3	5	5	1	1
5	3	1	4	2
4	2	2	4	5

1	2		5	
	5	3	1	4
3		5		1
5	3	1	4	2
4		2		5

1. Aucun nombre ne doit apparaître plus d'une fois dans chaque ligne et colonne.
2. Les carrés noirs ne doivent pas être adjacents.
3. Les carrés blancs doivent former une région contiguë unique

Singles - Programme linéaire (Part. 1)

Grille donnée : $a(i, j)$

Grille résultat : $x(i, j, k)$

$$(P) \begin{cases} \text{Maximiser} & \sum_{k=1}^n x(1, 1, k) \\ \text{s.c} & x(i, j, a(i, j)) + x(i, j, 0) = 1 \quad \forall i, j \in [1, n] \\ & \sum_{k=0}^n x(i, j, k) = 1 \quad \forall i, j \in [1, n] \\ & \sum_{i=1}^n x(i, j, k) \leq 1 \quad \forall j, k \in [1, n] \\ & \sum_{j=1}^n x(i, j, k) \leq 1 \quad \forall i, k \in [1, n] \\ & x(i, j, 0) + x(c, l, 0) \leq 1 \quad \forall \text{ cases adjacentes } (i, j) \text{ et } (c, l) \end{cases}$$

Singles - Programme linéaire (Part. 2)

3	2		3	
3	5	2		4
2	3	1	1	3
1	3	3	4	2
2	1	3	2	2

3	2	2		3
3	5	2	1	
2	3	1	1	3
1	3	3	4	2
2	1	3	2	2

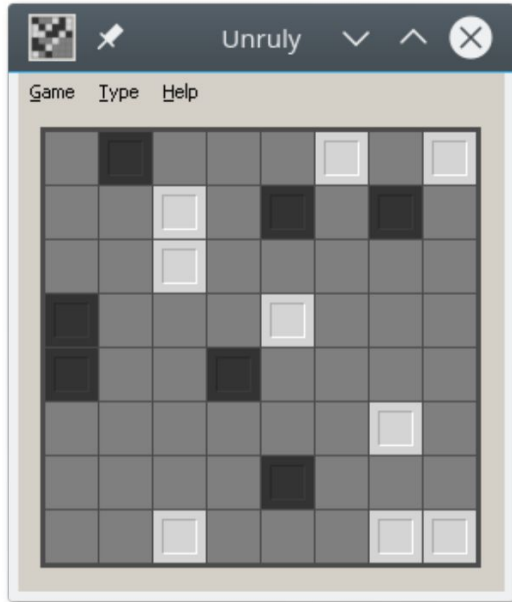
	2	2	3	3
3		2	1	4
2	3		1	3
1	3	3		2
2	1	3	2	

3	2	2	3	3
3	5	2	1	
2	3	1		3
1	3		4	2
2		3	2	2

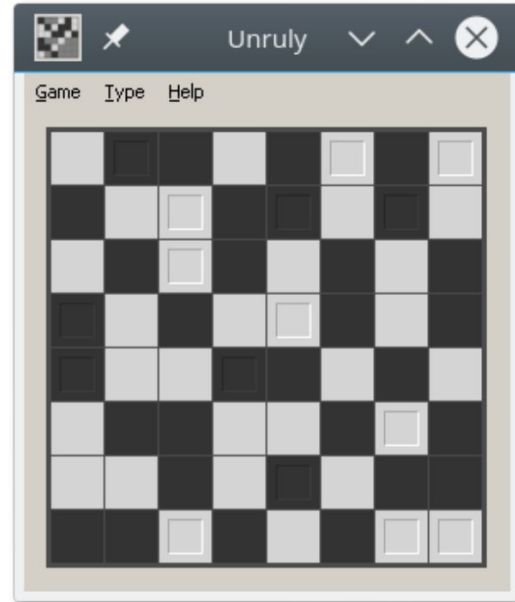
Singles - Difficultés rencontrés

- Génération d'instances aléatoires
- Application partielle des conditions
- Heuristique

Jeu 2 : Unruly



Exemple de grille initiale



Exemple de grille résolue

→ Remplir une grille en suivant 4 règles.

Génération d'instances

Parti pris : générer des instances "**potentiellement solvables**"

-> ie aucune règle enfreinte dans l'instance générée.

- pas d'alignement de 3 cases
- pas de lignes ou colonnes entièrement remplies

-> (pas suffisant pour assurer la solvabilité)

Pseudo Code:

```
generateInstance(n,m,d):  
  grid = grille_vide  
  while cases_remplies < cases_a_remplir & essais < essais_max:  
    case = case_aléatoire  
    couleur = couleur_aléatoire  
    if !colonne_remplie() & !ligne_remplie():  
      if !three_will_align():  
        grid[case] = couleur  
        cases_remplies +=1  
      essais +=1  
  end  
  return grid or return zeros
```



Examples:

```
t = generateInstance(10,10,0.2)
displayGrid(t)
```

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

Examples

```
t = generateInstance(10,10,0.6)
```



	-	1	-	-	1	2	1	1	-	1
	-	1	-	2	2	1	2	-	2	-
	-	-	2	2	-	1	-	1	1	-
	-	2	2	-	2	2	-	1	-	2
	2	1	-	-	2	-	2	1	1	-
	1	-	2	2	1	-	-	2	-	-
	1	2	-	1	-	2	-	-	2	1
	-	2	-	2	1	-	2	-	2	2
	2	-	1	-	-	1	2	1	1	1
	-	1	2	1	-	2	1	1	-	2

Examples

```
t = generateInstance(10,10,0.6)
```

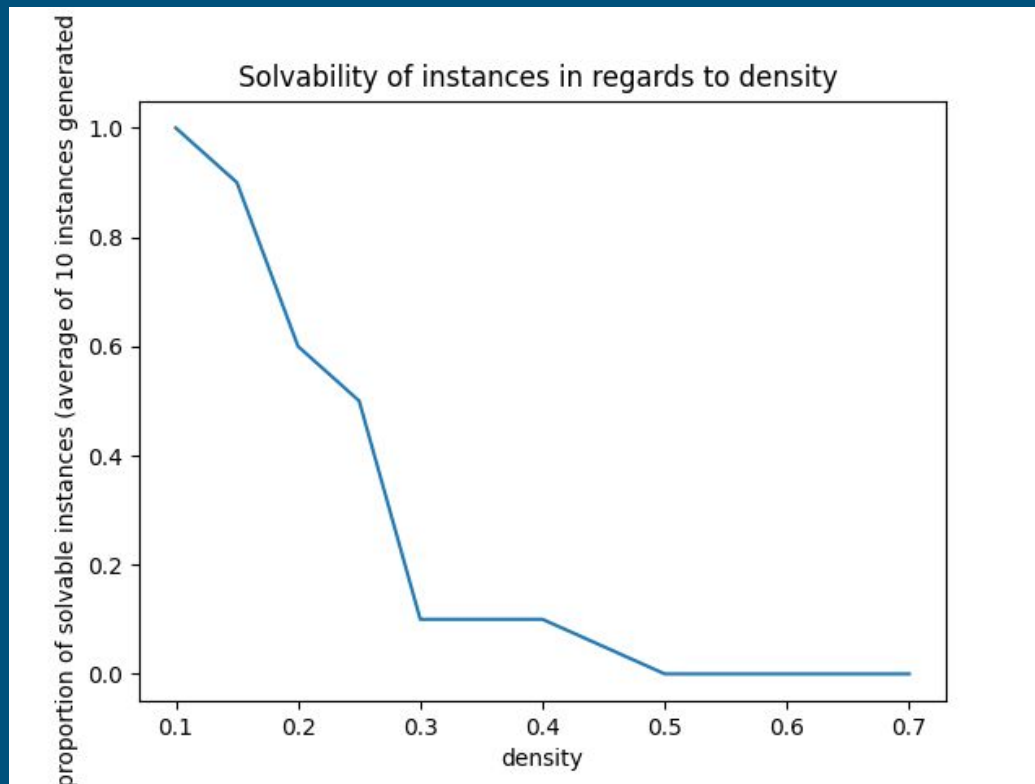
-	-	-	-	-	-	-	-	-	-	-	
	-	1	-	-	1	2	1	1	-	1	
	-	1	-	2	2	1	2	-	2	-	
	-	-	2	2	-	1	-	1	1	-	
	-	2	2	-	2	2	-	1	-	2	
	2	1	-	-	2	-	2	1	1	-	
	1	-	2	2	1	-	-	2	-	-	
	1	2	-	1	-	2	-	-	2	1	
	-	2	-	2	1	-	2	-	2	2	
	2	-	1	-	-	1	2	1	1	1	
	-	1	2	1	-	2	1	1	-	2	
-	-	-	-	-	-	-	-	-	-	-	-

Solvabilité en fonction de la densité

```
for density in [0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.5,0.6,0.7]  
  
    # Generate 10 instances  
    for instance in 1:10
```

puis on regarde l'optimalité

Graphique



Résolution :

```
### Contraintes

#chaque case initialisée doit être conservée
for i in 1:l
  for j in 1:c
    for k in 1:2
      if t[i,j]==k
        @constraint(m,x[i, j, k] == 1)
      end
    end
  end
end

#chaque case a une valeur unique
for i in 1:l
  for j in 1:c
    @constraint(m,sum(x[i, j, k] for k in 1:2) == 1)
  end
end

# pas plus de 2 cases identiques adjacentes horizontalement
for ligne in 1:l
  for colonne in 2:(c-1)
    for k in 1:2
      @constraint(m, sum(x[ligne, j, k] for j in (colonne-1):(colonne+1)) <= 2)
    end
  end
end

#pas plus de 2 cases identiques adjacentes verticalement
for colonne in 1:c
  for ligne in 2:(l-1)
    for k in 1:2
      @constraint(m, sum(x[i, colonne, k] for i in (ligne-1):(ligne+1)) <= 2)
    end
  end
end

#autant de cases blanches que de cases noires au sein d'une ligne
for ligne in 1:l
  @constraint(m, sum(x[ligne,j,1] for j in 1:c) == sum(x[ligne,j,2] for j in 1:c))
end

#autant de cases blanches que de cases noires au sein d'une colonne
for colonne in 1:c
  @constraint(m, sum(x[i,colonne, 1] for i in 1:l) == sum(x[i,colonne, 2] for i in 1:l))
end
```

Temps de résolution en fonction de la taille

taille 10 : 0.01s

taille 120 : 2.25s

taille 700 : 15s

Plus haute instance (densité 0.1) résolue : taille 120 en 2.25s

→ pourquoi ?

Idée d'heuristique

Si 2 alignement de 2 cases -> remplir les cases adjacentes de l'autre couleur

Si colonne ou ligne presque remplie -> remplir de la couleur manquante (parité)

Complexité de la génération d'instances:

```
generateInstance(n,m,d):  
  grid = grille_vide  
  while cases_remplies < cases_a_remplir & essais < essais_max:  
    case = case_aléatoire  
    couleur = couleur_aléatoire  
    if !colonne_remplie() & !ligne_remplie():  
      if !three_will_align():  
        grid[case] = couleur  
        cases_remplies +=1  
    essais +=1  
  end  
  return grid or return zeros
```

$N = n * m * d$

$N * n * m * 4$

$\rightarrow O(n^2 * m^2 * d * 4)$

pour $n = m = 10e3$, $d = 0.1$

complexité

$O(4 * 10e11) \sim O(10e11)$

$\rightarrow 15mn$ sans générer

Merci pour votre écoute