

# Résolution de jeux de logique

RO203 | Towers et Singles

---

Bruno MACEDO SANCHES, Cynthia LACROIX HERKENHOFF, Vladyslav HERASY-MENKO

ENSTA Paris | 05 mai 2022

1. Towers

2. Singles

# Towers

---

# Génération d'instances

Donnée: Taille  $n$  de la grille du jeu

Sortie: Matrice  $4 \times n$  avec le nombre de tours qui doivent être visibles à partir de chaque perspective

Représentation d'une instance:

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
|   |   | 3 | 2 | 1 | 2 | 2 |   |
| 2 |   |   |   |   |   |   | 2 |
| 2 |   |   |   |   |   |   | 4 |
| 3 |   |   |   |   |   |   | 2 |
| 4 |   |   |   |   |   |   | 1 |
| 1 |   |   |   |   |   |   | 3 |
|   | 1 | 4 | 2 | 3 | 2 |   |   |

→

$$\begin{bmatrix} 3 & 2 & 1 & 2 & 2 \\ 2 & 4 & 2 & 1 & 3 \\ 1 & 4 & 2 & 3 & 2 \\ 2 & 2 & 3 & 4 & 1 \end{bmatrix}$$

# Génération d'instances

Algorithme:

1. Initialiser un tableau  $n \times n$  rempli avec des zéros  
Pour chaque position  $\{(1, 1), (1, 2), \dots, (1, n), (2, 1), \dots, (n, n)\}$
2. Prendre tous les valeurs de 1 à  $n$  qui n'ont pas été mises dans la même ligne ou colonne
3. S'il existe de telles valeurs, prendre une aléatoirement et aller à la prochaine position
4. S'il n'y a pas de valeurs disponibles ou s'il a essayé toutes les valeurs possibles sans succès, retourner à la position antérieure et essayer une autre valeur
5. Retourner quand toutes les positions sont remplies
6. Au final, compter pour chaque perspective le nombre de tours visibles

# Génération d'instances

Exemples d'instances générées:

|           |  |  |  |  |   |
|-----------|--|--|--|--|---|
| 2 3 2 1 3 |  |  |  |  |   |
| -----     |  |  |  |  |   |
| 2         |  |  |  |  | 2 |
| 1         |  |  |  |  | 3 |
| 3         |  |  |  |  | 1 |
| 3         |  |  |  |  | 2 |
| 2         |  |  |  |  | 4 |
| -----     |  |  |  |  |   |
| 3 1 2 3 3 |  |  |  |  |   |

|                     |  |  |  |  |  |  |  |  |  |  |   |
|---------------------|--|--|--|--|--|--|--|--|--|--|---|
| 3 2 1 3 4 2 3 5 3 2 |  |  |  |  |  |  |  |  |  |  |   |
| -----               |  |  |  |  |  |  |  |  |  |  |   |
| 3                   |  |  |  |  |  |  |  |  |  |  | 2 |
| 4                   |  |  |  |  |  |  |  |  |  |  | 1 |
| 3                   |  |  |  |  |  |  |  |  |  |  | 4 |
| 2                   |  |  |  |  |  |  |  |  |  |  | 3 |
| 1                   |  |  |  |  |  |  |  |  |  |  | 6 |
| 3                   |  |  |  |  |  |  |  |  |  |  | 2 |
| 2                   |  |  |  |  |  |  |  |  |  |  | 4 |
| 4                   |  |  |  |  |  |  |  |  |  |  | 2 |
| 5                   |  |  |  |  |  |  |  |  |  |  | 4 |
| 6                   |  |  |  |  |  |  |  |  |  |  | 2 |
| -----               |  |  |  |  |  |  |  |  |  |  |   |
| 4 3 5 3 3 3 2 3 1 3 |  |  |  |  |  |  |  |  |  |  |   |

Modélisation en tant que PLNE

- Grille  $n \times n$
- Perspectives 1, 2, 3, 4

Pour la modélisation du jeu comme un PLNE, on définit les variables:

- $tours[i, j] = h, \quad h \in [1, n]$

- $\begin{cases} x_{ijh} = 1 & \text{si } tours[i, j] = h \text{ (càd la tour } ij \text{ a une hauteur } h) \\ x_{ijh} = 0 & \text{sinon} \end{cases}$

- $\begin{cases} v[i, j, p] = 1 & \text{si la } tours[i, j] \text{ est visible à partir de la perspective } p \\ v[i, j, p] = 0 & \text{sinon} \end{cases}$



Et aussi les variables:

$$\bullet \begin{cases} \text{limits}[p, i] = \sum_{j=1}^n v[i, j, p] & p \in \{1, 3\}, j \in [1, n] \\ \text{limits}[p, j] = \sum_{i=1}^n v[i, j, p] & p \in \{2, 4\}, i \in [1, n] \end{cases}$$

lignes :

$$\bullet \begin{cases} g_l[i, j, k] = 1 & \text{si } \text{tours}[i, j] > \text{tours}[i, k], \quad k \in [1, n] \\ g_l[i, j, k] = 0 & \text{sinon} \end{cases}$$

colonnes :

$$\bullet \begin{cases} g_c[i, j, k] = 1 & \text{si } \text{tours}[i, j] > \text{tours}[k, j], \quad k \in [1, n] \\ g_c[i, j, k] = 0 & \text{sinon} \end{cases}$$

Les contraintes du PLNE sont de nature suivante:

- Contraintes type sudoku
- Assurer les limites en tant que somme des visibilitées
- La première tour et celle de hauteur  $n$
- Assurer  $g_l$  et  $g_c$  à valeurs binaires bien définis
- Assurer  $v[i, j, p]$  à valeur binaire bien défini

Voici les contraintes:

1. Colonnes:  $\sum_{i=1}^n x_{ijh} = 1 \quad \forall j \forall h$
2. Lignes:  $\sum_{j=1}^n x_{ijh} = 1 \quad \forall i \forall h$
3.  $\sum_{h=1}^n x_{ijh} = 1 \quad \forall i \forall j$
4.  $\sum_{h=1}^n h \cdot x_{ijh} = \text{tours}[i, j] \quad \forall i \forall j$
5.  $\text{limits}[p, i] = \sum_{j=1}^n v[i, j, p], \quad p \in \{1, 3\}, j \in [1, n]$
6.  $\text{limits}[p, j] = \sum_{i=1}^n v[i, j, p], \quad p \in \{2, 4\}, i \in [1, n]$

7.  $v[1, j, 1] = 1, \quad j \in [1, n]$
8.  $v[i, n, 2] = 1, \quad i \in [1, n]$
9.  $v[n, j, 3] = 1, \quad j \in [1, n]$
10.  $v[i, 1, 4] = 1, \quad i \in [1, n]$
11.  $v[i, j, p] \geq x_{ijn}, \quad \forall p \in [1, 4]$
12.  $\text{tours}[i, k] - \text{tours}[i, j] \leq (1 - g_l[i, j, k]) \cdot n$
13.  $\text{tours}[i, k] - \text{tours}[i, j] \geq -g_l[i, j, k] \cdot n$
14.  $\text{tours}[k, j] - \text{tours}[i, j] \leq (1 - g_c[i, j, k]) \cdot n$
15.  $\text{tours}[k, j] - \text{tours}[i, j] \geq -g_c[i, j, k] \cdot n$

16.  $\sum_{k=1}^n g_c[i, j, k] \geq (i - 1) \cdot v[i, j, 1], \quad k \in [1, i - 1]$
17.  $\sum_{k=1}^n g_c[i, j, k] \leq (i - 2) + v[i, j, 1], \quad k \in [1, i - 1]$
18.  $\sum_{k=1}^n g_l[i, j, k] \geq (i - 1) \cdot v[i, j, 2], \quad k \in [1, i - 1]$
19.  $\sum_{k=1}^n g_l[i, j, k] \leq (i - 2) + v[i, j, 2], \quad k \in [1, i - 1]$
20.  $\sum_{k=1}^n g_c[i, j, k] \geq (i - 1) \cdot v[i, j, 3], \quad k \in [1, i - 1]$
21.  $\sum_{k=1}^n g_c[i, j, k] \leq (i - 2) + v[i, j, 3], \quad k \in [1, i - 1]$
22.  $\sum_{k=1}^n g_l[i, j, k] \geq (i - 1) \cdot v[i, j, 4], \quad k \in [1, i - 1]$
23.  $\sum_{k=1}^n g_l[i, j, k] \leq (i - 2) + v[i, j, 4], \quad k \in [1, i - 1]$

|   | 1 | 3 | 2 | 4 | 3 | 2 | 3 | 3 | 3 |   |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 |   |   |   |   |   |   |   |   |   | 4 |
| 4 |   |   |   |   |   |   |   |   |   | 2 |
| 3 |   |   |   |   |   |   |   |   |   | 2 |
| 2 |   |   |   |   |   |   |   |   |   | 4 |
| 6 |   |   |   |   |   |   |   |   |   | 1 |
| 5 |   |   |   |   |   |   |   |   |   | 2 |
| 3 |   |   |   |   |   |   |   |   |   | 3 |
| 4 |   |   |   |   |   |   |   |   |   | 3 |
| 2 |   |   |   |   |   |   |   |   |   | 3 |
|   | 5 | 1 | 3 | 2 | 4 | 4 | 2 | 3 | 3 |   |

(a) Instance du jeu

|   | 1 | 3 | 2 | 4 | 3 | 2 | 3 | 3 | 3 |   |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 9 | 6 | 8 | 5 | 7 | 1 | 3 | 2 | 4 | 4 |
| 4 | 5 | 1 | 3 | 7 | 8 | 9 | 2 | 4 | 6 | 2 |
| 3 | 7 | 8 | 6 | 3 | 4 | 5 | 1 | 9 | 2 | 2 |
| 2 | 8 | 5 | 2 | 1 | 9 | 4 | 7 | 6 | 3 | 4 |
| 6 | 1 | 3 | 4 | 2 | 6 | 8 | 5 | 7 | 9 | 1 |
| 5 | 4 | 2 | 5 | 6 | 1 | 7 | 9 | 3 | 8 | 2 |
| 3 | 6 | 7 | 9 | 8 | 2 | 3 | 4 | 1 | 5 | 3 |
| 4 | 3 | 4 | 7 | 9 | 5 | 2 | 6 | 8 | 1 | 3 |
| 2 | 2 | 9 | 1 | 4 | 3 | 6 | 8 | 5 | 7 | 3 |
|   | 5 | 1 | 3 | 2 | 4 | 4 | 2 | 3 | 3 |   |

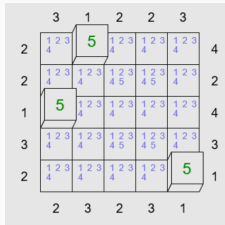
(b) Solution

Figure 1: Instance et solution du jeu

# Heuristique

Idée: Trouver un algorithme qui cherche les solution parmi les hauteurs possibles pour chaque tour

- Éliminer des possibilités
- Essayer avec les possibilités existantes jusqu'à ce qu'on trouve une solution



Algorithme:

1. Initialiser un tableau  $M_{n \times n \times n}$  binaire où  $M_{ijk} = 1$  si la hauteur  $k$  est possible dans la tour  $(ij)$ . Initialiser le tableau avec toutes les valeurs égales à 1
2. Traiter les valeurs triviales 1 et  $n$ , en éliminant les possibilités au maximum
3. Prendre une position considérée bonne (unique tour pour laquelle une hauteur est possible ou peu de hauteurs sont possibles)
4. Vérifier si la valeur respecte les contraintes de visibilité
5. Si oui, l'essayer et retourner à l'étape 2
6. Sinon, vérifier une autre valeur et répéter
7. S'il n'y a plus de valeurs, retourner à la position antérieure et essayer d'autres valeurs



Il est possible d'améliorer l'algorithme en implémentant des règles pour éliminer des possibilités

Implémentées:

- Éliminer les valeurs déjà utilisées dans la ligne ou colonne
- Si une hauteur n'est possible que dans une tour, la seule possibilité pour cette tour sera cette hauteur

Pas implémentées mais possibles:

- Si deux valeurs ne sont possibles que dans deux tours, ces tours auront juste ces possibilités
- Limitation de hauteur: i.e  $n = 5$
- Si la valeur de visibilité est 3, on est sûr de ne pas avoir 4 comme la hauteur de la première tour

# Singles

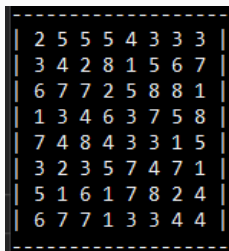
---

# Génération d'instances

Donnée: Taille  $n$  de la grille du jeu et densité  $d \in [0, 1]$  représentant la densité de nombres répétés dans le tableau

Sortie: Matrice  $n \times n$  avec  $\lfloor n^2 \times d \rfloor$  nombres répétés

Exemple d'instance générée par le programme:



|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| 2 | 5 | 5 | 5 | 4 | 3 | 3 | 3 |
| 3 | 4 | 2 | 8 | 1 | 5 | 6 | 7 |
| 6 | 7 | 7 | 2 | 5 | 8 | 8 | 1 |
| 1 | 3 | 4 | 6 | 3 | 7 | 5 | 8 |
| 7 | 4 | 8 | 4 | 3 | 3 | 1 | 5 |
| 3 | 2 | 3 | 5 | 7 | 4 | 7 | 1 |
| 5 | 1 | 6 | 1 | 7 | 8 | 2 | 4 |
| 6 | 7 | 7 | 1 | 3 | 3 | 4 | 4 |

Figure 2:  $n = 8, d = 0.3$

Algorithme:

1. Générer un tableau rempli par des nombres uniques dans chaque colonne et ligne, comme en towers.
2. Déterminer le nombre de cases à masquer
3. Déterminer les cases disponibles (initialement, le tableau entier)
4. Tandis qu'il n'a pas masqué suffisamment de cases et il y a encore des cases disponibles, sélectionner aléatoirement une case et la retirer de l'ensemble de cases disponibles
5. S'il est possible de masquer cette case (ne viole pas le contrainte de connexité), masquer cette case et retirer les cases adjacentes de l'ensemble de cases disponibles

6. S'il n'est pas possible, passer à l'étape 4.
7. S'il a pas trouvé une solution, retourner à la position sélectionnée antérieurement
8. S'il a trouvé un tableau de masques possible, pour chaque position masqué, sélectionner un nombre aléatoire présent dans la même colonne ou ligne et le mettre dans la positions masqué

Le tableau sera le problème et le tableau de masques la solution attendu.

Variables représentant les cases de la grille masquées :

$$\bullet \begin{cases} x_{ij} = 1 & \text{si la case est masquée} \\ x_{ij} = 0 & \text{sinon} \end{cases}$$

Paramètres du jeu qui représentent la valeur d'une case donnée :

$$\bullet \begin{cases} valeur_{ijk} = 1 & \text{si la valeur de case}(i,j) \text{ est égale à } k \\ valeur_{ijk} = 0 & \text{sinon} \end{cases}$$

Non-répétition des valeurs sur une ligne :

$$\forall k \forall i \quad \sum_{j=1}^n \text{valeur}_{ijk} \cdot (1 - x_{ij}) \leq 1 \quad (1)$$

Ou sur une colonne :

$$\forall k \forall j \quad \sum_{i=1}^n \text{valeur}_{ijk} \cdot (1 - x_{ij}) \leq 1 \quad (2)$$

Non-adjacence :

$$\forall i \in \{2 \dots n\} \forall j \quad x_{ij} + x_{(i-1)j} \leq 1 \quad (3)$$

$$\forall i \in \{1 \dots n-1\} \forall j \quad x_{ij} + x_{(i+1)j} \leq 1 \quad (4)$$

$$\forall i \forall j \in \{2 \dots n\} \quad x_{ij} + x_{i(j-1)} \leq 1 \quad (5)$$

$$\forall i \forall j \in \{1 \dots n-1\} \quad x_{ij} + x_{i(j+1)} \leq 1 \quad (6)$$



Du à la complexité de la contrainte de connexité, on utilisera une approche itérative dont l'algorithme est décrit ci-dessous :

1. Résoudre le programme linéaire :

$$\max(z) = 1$$

s.c. (1), (2), (3), (4), (5) et (6)

2. Vérifier si la solution obtenue à l'étape 1 est connexe (à l'aide de la théorie des graphes).
3. Si la solution est connexe, on s'arrête. Sinon, aller à 4.
4. Identifier toutes les cases masquées de la solution obtenue (qui n'est pas connexe). On note cet ensemble  $g$  :

$$g \equiv \{(x, i) | x_{ij} = 1\}$$

5. Compter le nombre des cases masquées. Notons ce nombre  $k$  :

$$k = \sum_{i,j} x_{ij}$$

6. Interdire la solution non-connexe composée des cases de l'ensemble  $g$ . C'est-à-dire, ajouter une nouvelle contrainte :

$$\sum_{(i,j) \in g} x_{ij} \leq k - 1$$

Finalement aller à 1 en y ajoutant la contrainte ci-dessus.

|    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|
| 11 | 12 | 7  | 3  | 3  | 10 | 8  | 5  | 9  | 6  | 10 | 8  |
| 3  | 4  | 11 | 3  | 1  | 5  | 10 | 10 | 6  | 6  | 12 | 11 |
| 5  | 10 | 1  | 8  | 2  | 8  | 1  | 4  | 7  | 11 | 9  | 12 |
| 9  | 9  | 6  | 8  | 7  | 10 | 1  | 2  | 3  | 3  | 10 | 6  |
| 9  | 3  | 5  | 10 | 3  | 12 | 5  | 1  | 11 | 1  | 12 | 9  |
| 7  | 9  | 10 | 12 | 8  | 9  | 4  | 11 | 3  | 3  | 2  | 1  |
| 4  | 12 | 9  | 4  | 5  | 11 | 12 | 10 | 1  | 8  | 11 | 7  |
| 8  | 7  | 5  | 9  | 1  | 3  | 11 | 2  | 8  | 2  | 1  | 3  |
| 1  | 10 | 3  | 9  | 12 | 7  | 9  | 8  | 1  | 10 | 11 | 6  |
| 8  | 1  | 11 | 7  | 11 | 2  | 5  | 12 | 10 | 1  | 5  | 3  |
| 1  | 5  | 4  | 12 | 6  | 1  | 8  | 3  | 12 | 9  | 2  | 2  |
| 3  | 8  | 6  | 4  | 11 | 9  | 9  | 1  | 11 | 5  | 6  | 1  |

(a) Instance du jeu

|    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|
| 11 | 12 | 7  |    | 3  | 10 |    | 5  | 9  | 6  |    | 8  |
|    | 4  |    | 3  | 1  | 5  | 10 |    | 6  |    | 12 | 11 |
| 5  | 10 | 1  |    | 2  | 8  |    | 4  | 7  | 11 | 9  | 12 |
| 9  |    | 6  | 8  | 7  |    | 1  | 2  | 3  |    | 10 |    |
|    | 3  |    | 10 |    | 12 | 5  |    | 11 | 1  |    | 9  |
| 7  | 9  | 10 | 12 | 8  |    | 4  | 11 |    | 3  | 2  | 1  |
| 4  |    | 9  |    | 5  | 11 | 12 | 10 | 1  | 8  |    | 7  |
|    | 7  | 5  | 9  |    | 3  | 11 |    | 8  | 2  | 1  |    |
| 1  |    | 3  |    | 12 | 7  | 9  | 8  |    | 10 | 11 | 6  |
| 8  | 1  | 11 | 7  |    | 2  |    | 12 | 10 |    | 5  | 3  |
|    | 5  | 4  |    | 6  | 1  | 8  | 3  | 12 | 9  |    | 2  |
| 3  | 8  |    | 4  | 11 | 9  |    | 1  |    | 5  | 6  |    |

(b) Solution

Figure 3: Instance et solution du jeu *singles*

On note  $E$ , l'ensemble des cases à même valeur.

On utilisera une version simplifiée de branch-and-bound.

Notamment, on masquera des combinaisons des cases de  $E$  en branchant si on a en même temps les deux conditions suivantes:

- Contraintes de non-adjacence et de connexité sont vérifiées
- Contraintes de non-répétition des valeurs ne sont pas vérifiées

On coupe la branche si l'une des contraintes de non-adjacence ou de connexité n'est pas vérifiée.

Si toutes les contraintes sont vérifiées, on a notre solution et on s'arrête.

## Conclusion

- Comparaison: simplexe vs heuristique
- Complexité et taille du jeu
- Génération d'instances