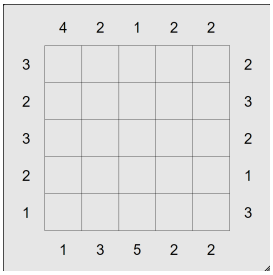
# RO203

# Résolution de jeux - Pegs & Tower

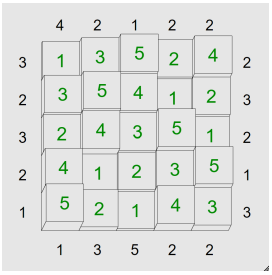
Valentin Quoniam-Barre - Axel Dumont

Avril 2023

# Jeu n°1 : Towers




(a) Nouveau jeu




(b) Jeu résolu

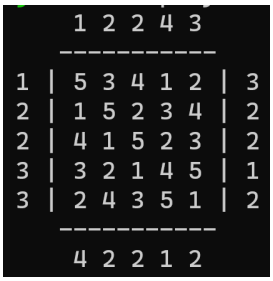
Figure 1 – Towers game

Le jeu consiste à remplir une grille selon les informations situées en haut, en bas, à gauche et à droite. On doit à la fin du jeu retrouver dans chaque colonne et chaque ligne les nombres de 1 à $n$ sans doublons. Ce nombre correspond à la taille du bâtiment placé dans cette case. Les vecteurs informations contiennent le nombre de bâtiments visibles depuis la position où il est.

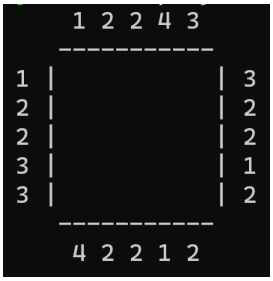
## 1 Modélisation

Pour la modélisation du jeu, nous avons opté pour la création d'une grille $n \times n$ et quatre vecteurs haut, bas, gauche, droite de taille $n$ qui contiennent l'information du nombre de tours visibles.

## 2 Génération d'instance




(a) Génération d'une grille




(b) Nouvelle instance

Figure 2 – Logique de génération

Dans la même logique que le Sudoku, on va d'abord générer une grille $n \times n$ puis calculer les vecteurs informations (haut, bas, gauche, droite) pour ensuite enlever la grille.

Pour générer la grille dans un premier temps, on remplit ligne par ligne en utilisant des nombres aléatoires et en vérifiant que la grille remplit les critères (sur les lignes et les colonnes). On recommence de zéro s'il s'avère à la fin que la grille ne correspond pas.

Par la suite on a juste à créer une fonction booléenne qui est vraie si la tour en $(i, j)$ est visible ou non pour générer les vecteurs informations.

# 3 Programmation linéaire

On utilise ici la méthode cplex pour résoudre ce problème

Pour ce faire on va définir les variables suivantes :

$$\forall (i,j,k) \in \{1,...,n\} \quad x(i,j,k) = \begin{cases} 1 & \text{si la case } (i,j) \text{ contient } k \\ 0 & \text{sinon} \end{cases}$$

$$\forall (i,j) \in \{1,...,n\} \quad y_h(i,j) = \begin{cases} 1 & \text{si la case } (i,j) \text{ est visible depuis le haut} \\ 0 & \text{sinon} \end{cases}$$

De même on définit $y_b(i,j)$, $y_g(i,j)$, $y_d(i,j)$ pour savoir si $(i,j)$ est visible par le bas, la gauche et la droite.

Maintenant, on peut définir les contraintes. Déjà, on peut dire qu'il ne peut y avoir qu'un chiffre par case, et qu'on a une seule occurrence d'un chiffre sur une colonne et une ligne. Ceci se traduit par :

$$\forall (i,j) \in \{1,...,n\}, \quad \sum_{k=1}^{n} x(i,j,k) = 1$$

$$\forall (i,k) \in \{1,...,n\}, \quad \sum_{j=1}^{n} x(i,j,k) = 1$$

$$\forall (j,k) \in \{1,...,n\}, \quad \sum_{i=1}^{n} x(i,j,k) = 1$$

Ensuite, le nombre de tours visibles doit être égal au nombre dans le vecteur information. On prend l'exemple du haut :

$$\forall j \in \{1,...,n\}, \quad \sum_{i=1}^{n} y_h(i,j) = haut(j)$$

Ensuite on s'intéresse à la somme :

$$\forall (i,j,k) \in \{1,...,n\} \quad \sum_{i_2<i} \sum_{k_2>k} x(i_2,j,k_2)$$

Cette somme correspond au nombre de tours plus grandes que $k$ au dessus dans la même colonne que $(i,j)$ (soit $j$) et vaut au maximum n, donc on peut normaliser et en déduire les contraintes suivantes :

$$\forall (i,j,k) \in \{1,...,n\} \quad y_h(i,j) \leq 1 - \frac{\sum_{i_2<i} \sum_{k_2>k} x(i_2,j,k_2)}{n} + 1 - x(i,j,k)$$

On rajoute le terme en $1 - x(i,j,k)$ pour éviter de fixer $y_h(i,j)$ quand $x(i,j,k) = 0$

De même on a la contrainte :

$$\forall (i,j,k) \in \{1,...,n\} \quad y_h(i,j) \leq 1 - \sum_{i_2<i} \sum_{k_2>k} x(i_2,j,k_2) - n(1 - x(i,j,k))$$

On peut étendre ce raisonnement avec les autres variables $y$.

# 4 Résultats

On résout avec des instances jusqu'à la taille 6. Après expérience, on voit qu'on arrive pas à trouver de résultats pour des tailles supérieures à 10.

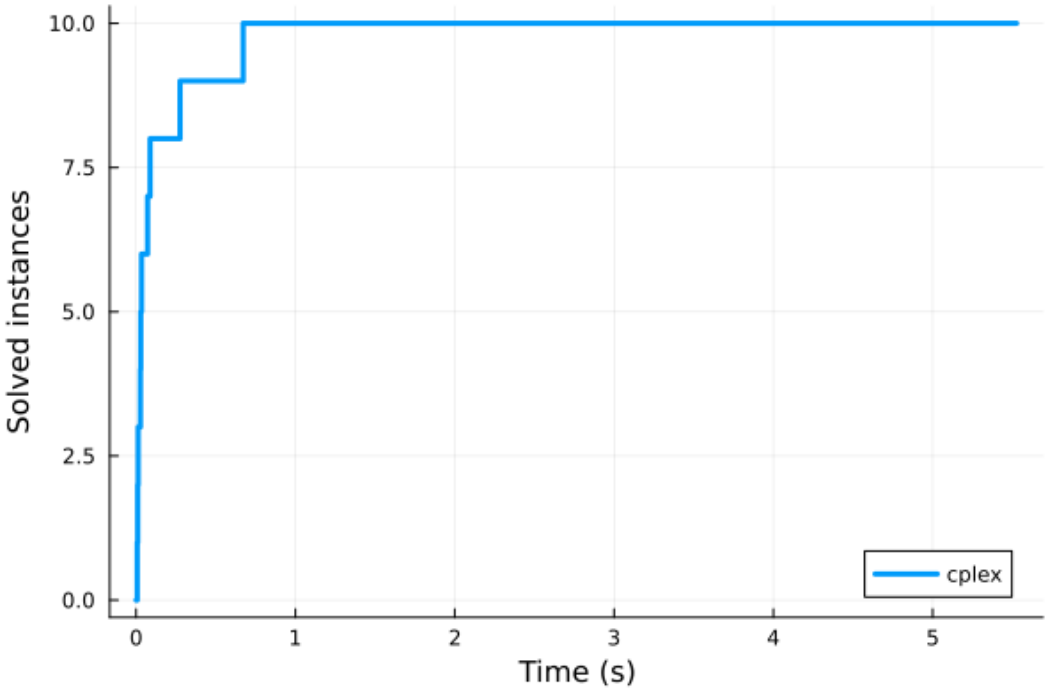



Figure 3 – Résultats

| | cplex | |
|---|---|---|
| **Instance** | **Temps (s)** | **Optimal ?** |
| instance_t3_1.txt | 5.53 | × |
| instance_t3_2.txt | 0.01 | × |
| instance_t4_1.txt | 0.01 | × |
| instance_t4_2.txt | 0.01 | × |
| instance_t5_1.txt | 0.03 | × |
| instance_t5_2.txt | 0.03 | × |
| instance_t6_1.txt | 0.09 | × |
| instance_t6_2.txt | 0.07 | × |
| instance_t7_1.txt | 0.67 | × |
| instance_t7_2.txt | 0.28 | × |
| instance_test.txt | 0.03 | × |

# Jeu n°2 : Pegs


(a) Nouveau jeu (b) Jeu résolu

FIGURE 4 – Pegs game

Le jeu consiste à réduire au maximum le nombre de pions sur la grille. Pour cela, les pions fonctionnent comme au jeu de dames : un pion peut sauter par-dessus un autre en l'éliminant. Une étape de résolution consiste donc à faire sauter un pion au dessus d'un autre. Il faut pour cela que le pion en question aie un pion adjacent et une case libre apres ce dernier. On peut dors et déjà noter qu'une résolution prend au maximum $s$ étapes, où $s$ est le nombre de pions initiaux.

## 1 Modélisation et génération d'instances

Pour la modélisation du jeu, nous avons opté pour la création d'une grille $n \times n$, appelée *grid*, telle que :

$$\forall (i,j) \in \{1,...,n\} \quad grid[i,j] = \begin{cases} 1 & \text{si la case } (i,j) \text{ est occupée} \\ 0 & \text{si la case } (i,j) \text{ est vide} \\ 2 & \text{si la case } (i,j) \text{ est hors de la grille} \end{cases}$$

Pour représenter plus lisiblement une grille, la fonction *DisplayGrid(grid)* va afficher "o" si une case est vide, "■" si elle est pleine, et va ignorer les cases marquées avec un 2.


(a) Instance de base (b) Instance plus lisible

FIGURE 5 – Instances du jeu Pegs

Pour générer une instance, c'est assez simple. On génère une matrice $n \times n$ et une densité aléatoire entre 0 et 1. Chaque case de la matrice hors-grille vaut 2, et chaque autre case se voit attribuer une valeur aléatoire entre 0 et 1. Toute case dont la valeur est au-dessus de la densité devient un 1, et toutes les autres cases deviennent un 0.

## 2 Programmation linéaire

On utilise ici la méthode cplex pour résoudre ce problème

Pour ce faire on va définir les variables suivantes :

$$\forall (i,j) \in \{1,...,n\}, t \in 1,...,s \quad bState[i,j,t] = \begin{cases} 1 & \text{si la case } (i,j) \text{ est occupée à l'étape } t \\ 0 & \text{sinon} \end{cases}$$

$$\forall (i,j) \in \{1,...,n\}, t \in \{1,...,s\}, d \in 1,...,4, \quad M[i,j,t,d] = \begin{cases} 1 & \text{si le pion de case } (i,j) \text{ bouge dans la direction } d \text{ à l'étape } t \\ 0 & \text{sinon} \end{cases}$$

Avec comme directions : d = 1 : Nord | d = 2 : Sud | d = 3 : Est | d = 4 : Ouest

Attention : n n'est plus la taille de la grille, mais est la taille + 4. En effet, pour étudier la faisabilité d'un saut, il faut étudier les cases 2 indices plus loin dans toutes les directions. On est alors obligé d'avoir un incrément de 4 pour considérer toutes les cases de la grille et ne pas en ignorer.

On peut maintenant définir les contraintes.

Commençons par celles qui définissent le déplacement dans les 4 directions :
Pour qu'un saut puisse se faire, il faut :

1. Que la case choisie contienne un pion : contrainte (1)
2. Que la case adjacente dans la direction du saut soit occupée : contraintes (2) - (4) - (6) - (8)
3. Que la case d'arrivée du saut soit vide : contraintes (1) - (5) - (7) - (9)

$$\begin{aligned}
\forall d \in 1,...,4, \quad & M[i,j,t,d] \leq bState[i,j,t] && (1)\\
& M[i,j,t,1] \leq bState[i,j-1,t] && (2)\\
& M[i,j,t,1] \leq (1 - bState[i,j-2,t]) && (3)\\
& M[i,j,t,2] \leq bState[i,j+1,t] && (4)\\
& M[i,j,t,2] \leq (1 - bState[i,j+2,t]) && (5)\\
& M[i,j,t,3] \leq bState[i+1,j,t] && (6)\\
& M[i,j,t,3] \leq (1 - bState[i+2,j,t]) && (7)\\
& M[i,j,t,4] \leq bState[i-1,j,t] && (8)\\
& M[i,j,t,4] \leq (1 - bState[i-2,j,t]) && (9)
\end{aligned}$$

avec $(i,j) \in \{1,...,n\}$, $t \in \{1,...,s-1\}$ pour ces équations

(Par souci de lisibilité, nous n'avons pas détaille évidemment, que $i > 2$ quand on étudie la case $i-2$, $j < n$ quand on étudie la case $j+1$, etc...)

Ensuite, il faut préciser qu'il y a au maximum un saut par étape de résolution :

$$\forall t \in \{1,...,(s-1)\}, \quad \sum_{i,j=1}^{n} M[i,j,t,1] + M[i,j,t,2] + M[i,j,t,3] + M[i,j,t,4]) \leq 1 \qquad (10)$$

Ensuite, on définit la principale contrainte : celle qui gère la transition entre l'étape $t$ et l'étape $t+1$. Il faut que la différence de pions entre 2 étapes soit égale à la somme de tous les mouvements entre ces 2 étapes (en enlevant les mouvement des trous sur lesquelles un pion atterrit).

$$\begin{aligned}\forall (i,j) \in \{3,...,n-2\}, t \in 1,...,s-1, \quad bState[i,j,t] - bState[i,j,t+1] = &\sum_{d=1}^{4} M[i,j,t,d]) \\ &+ M[i-1,j,t,3] - M[i-2,j,t,3] \\ &+ M[i+1,j,t,4] - M[i+2,j,t,4] \\ &+ M[i,j-1,t,2] - M[i,j-2,t,2] \\ &+ M[i,j+1,t,1] - M[i,j+2,t,1]\end{aligned}$$

Enfin, on définit comme objectif de minimiser le nombre de pions sur la grille. Cet objectif est incomplet, mais il est impossible de faire mieux en faisant uniquement de la programmation linéaire.

$$objectif = \min(\sum_{i,j=1}^{n} bState[i,j,s]) \tag{11}$$

# 3 Résultats

On résout des instances de taille 7 et 16 uniquement pour des raisons de temps de calcul. Il est possible de résoudre des instances plus grandes avec des temps de calcul allant jusqu'à 3 minutes.

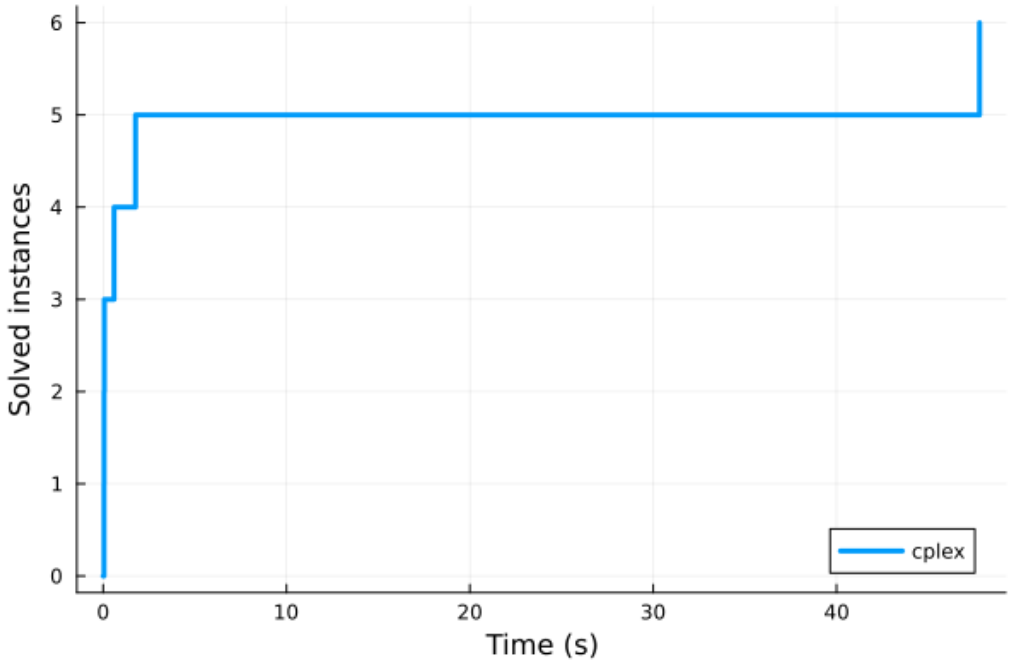



FIGURE 6 – Résultats pour le Pegs

| | cplex | |
|---|---|---|
| **Instance** | **Temps (s)** | **Optimal ?** |
| instance_1.txt | 61.34 | |
| instance_10.txt | 1.77 | × |
| instance_2.txt | 60.66 | |
| instance_3.txt | 47.8 | × |
| instance_4.txt | 0.05 | × |
| instance_5.txt | 0.05 | × |
| instance_6.txt | 0.07 | × |
| instance_7.txt | 0.6 | × |
| instance_8.txt | 60.24 | |
| instance_9.txt | 60.36 | |