RO203 - Towers & Pegs

LE SAUX Loup - GERMAIN Antoine

2 Mai 2023

Sommaire

Le jeu Towers

Règles

- Pas de doublons sur une ligne ;
- Pas de doublons sur une colonne ;
- Chaque indice autour du bord compte le nombre de tours visibles lorsque l'on regarde la grille dans cette direction.

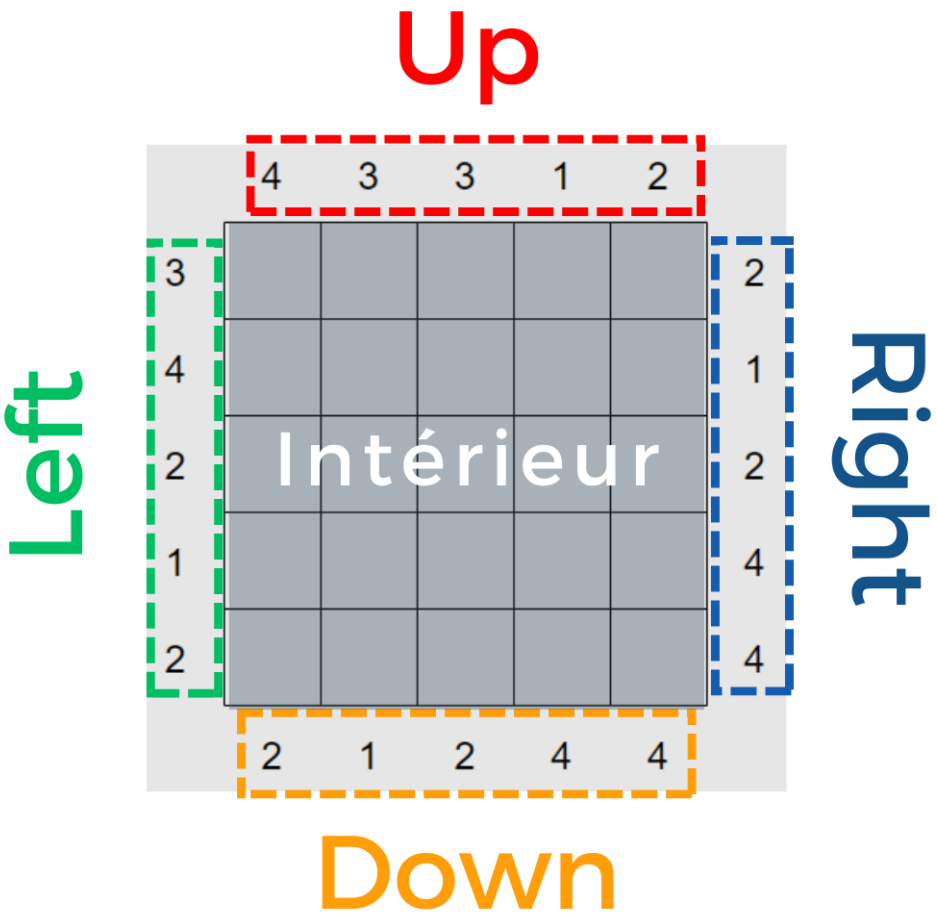


Figure 1 – Modélisation du problème

Variables et objectif

Variables

$$x_{i,j,k} := \begin{cases} 1 & \text{si la tour de hauteur } k \text{ est dans la case } (i,j) \\ 0 & \text{sinon} \end{cases}$$

$$yu_{i,j} := \begin{cases} 1 & \text{si la tour contenue dans la case } (i,j) \text{ est visible depuis Up} \\ 0 & \text{sinon} \end{cases}$$

$\Longrightarrow$ idem $yl_{i,j}, \quad yr_{i,j}, \quad yd_{i,j}$.

Fonction objectif

$$\max \sum_{k=1}^{n} x_{1,1,k}.$$

Contraintes

Contraintes de base

- Sur une colonne les tours sont toutes de hauteurs différentes :

$$\forall (i,k) \in \{1,\dots,n\}^2 \quad \sum_{j=1}^{n} x_{i,j,k} = 1.$$

- Sur une ligne les tours sont toutes de hauteurs différentes :

$$\forall (k,j) \in \{1,\dots,n\}^2 \quad \sum_{i=1}^{n} x_{i,j,k} = 1.$$

- Chaque case doit contenir une tour :

$$\forall (i,j) \in \{1,\dots,n\}^2 \quad \sum_{k=1}^{n} x_{i,j,k} = 1.$$

Contraintes

Contraintes de visibilité

- Nombre de tours visibles depuis Up :

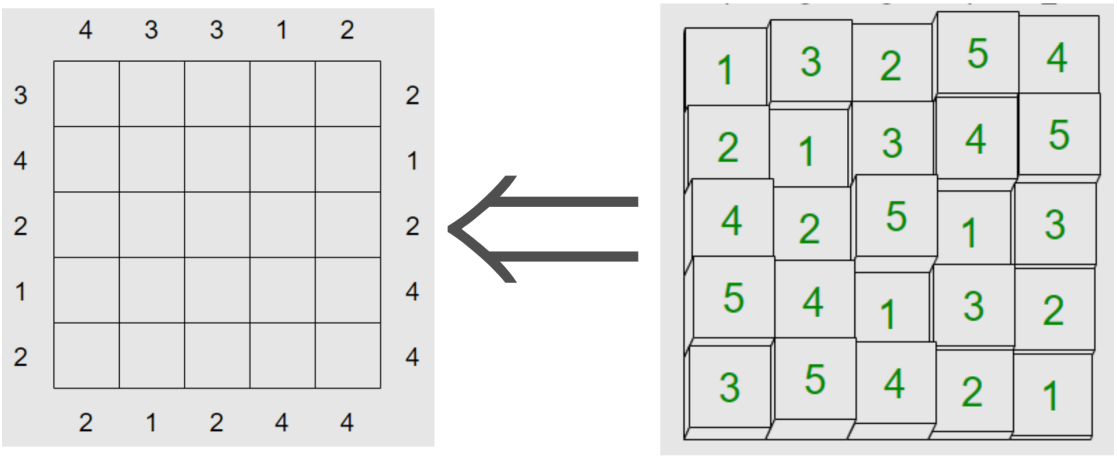
$$\forall j \in \{1, \dots, n\} \quad \sum_{i=1}^{n} yu_{i,j} = \mathsf{Up}[j].$$

$$\forall (i,j,k) \in [\![1,n]\!]^3 \quad yu_{i,j} \leq 1 - \frac{1}{n} \sum_{k'=k+1}^{n} \sum_{i'=1}^{i-1} x_{i',j,k'} + (1 - x_{i,j,k})$$

$$\forall (i,j,k) \in [\![1,n]\!]^3 \quad yu_{i,j} \geq 1 - \sum_{k'=k+1}^{n} \sum_{i'=1}^{i-1} x_{i',j,k'} - n(1 - x_{i,j,k})$$

$\Longrightarrow$ idem $yl_{i,j}, \quad yr_{i,j}, \quad yd_{i,j}.$

Génération des instances




```
      2 2 2 3 1
    -------------
3 | 3 2 4 1 5 | 1
2 | 2 5 1 4 3 | 3
1 | 5 1 3 2 4 | 2
2 | 4 3 2 5 1 | 2
3 | 1 4 5 3 2 | 3
    -------------
      3 2 1 2 3
```

- **Étape 1** : Générer des grilles complètes admissibles avec la fonction `generateGrid` et l'outil `is_valuable`.
- **Étape 2** : Générer les vecteurs Up, Down, Left, Right associés avec `generateVectors`.

Résultats

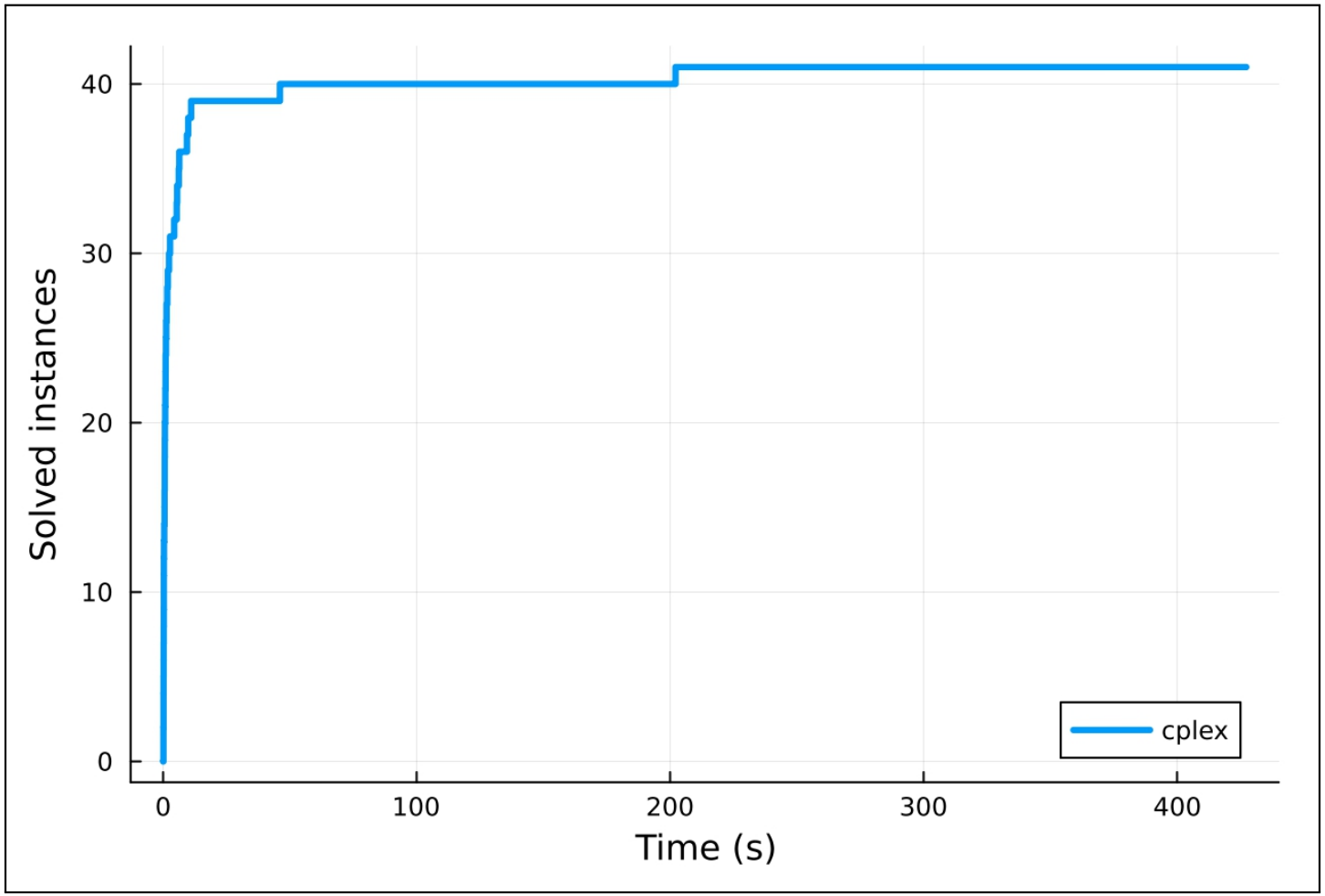


Figure 2 – Graphique du nombre d'instances résolues, par ordre croissant de difficulté, en fonction du temps de résolution pour 40 instances admissibles générées (10 pour chaque taille de grille $n \in \{5, 6, 7, 8\}$)

Bilan Towers

Complexité : résolution d'une grille

- Complexité temporelle : *Non polynomiale*

Points clés :

- Un cplex valide
- Génération, affichage et résolution complets
- Idée d'heuristique ?

Sommaire

Le jeu Pegs

Règles

- Minimiser le nombre de pions sur la grille ;
- A chaque étape, un pion peut être éliminé en sautant par-dessus un autre.

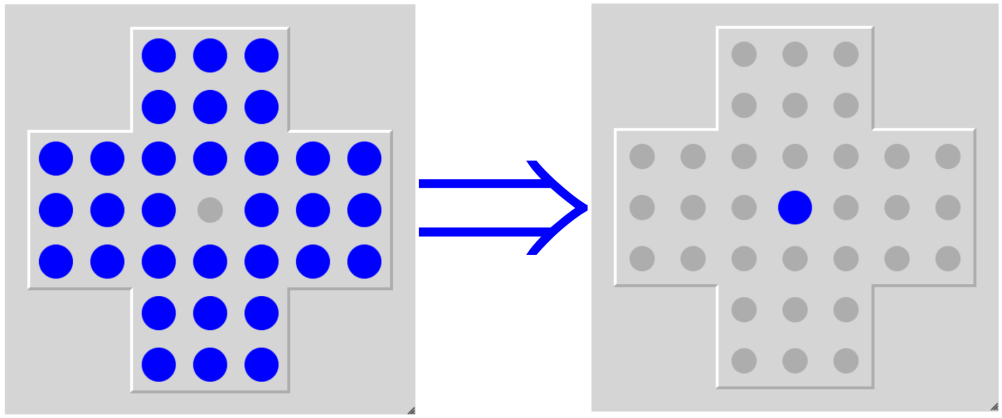


Figure 3 – Illustration du plateau en croix

Modélisation

Définition : grille G

Une grille G est l'état 0 de la partie.

- $G[i, j] = 0$: case vide
- $G[i, j] = 1$: pion
- $G[i, j] = 2$: hors-jeu

Aspect temporel d'une partie

Une partie dure n étapes avec n le nombre initial de pions

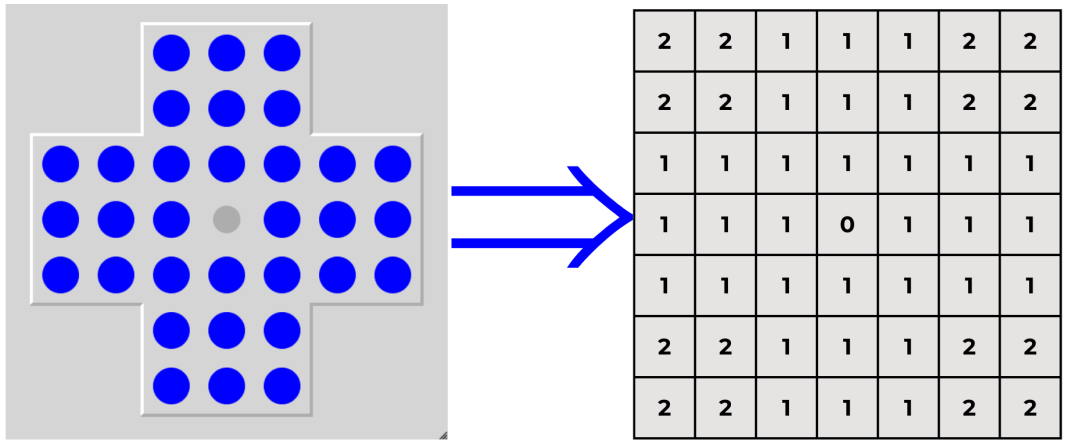


Figure 4 – Modélisation

Variables et objectif

Variables à l'état t du jeu

$$x_{i,j,t} := \begin{cases} 1 & \text{si un pion est présent dans la case } (i,j) \text{ à l'étape } t \\ 0 & \text{si la case } (i,j) \text{ ne contient pas de pion à l'étape } t \end{cases}$$

$d : 1 = \text{"Nord"}, 2 = \text{"Sud"}, 3 = \text{"Ouest"}, 4 = \text{"Est"}$

$$y_{i,j,t,d} := \begin{cases} 1 & \text{si le pion en } (i,j) \text{ admet un déplacement possible vers } d \\ 0 & \text{sinon} \end{cases}$$

Fonction objectif

L'objectif est de minimiser le nombre de pions sur la grille à l'étape n :

$$\min \sum_{i=1}^{m} \sum_{j=1}^{m} x_{i,j,n}. \tag{1}$$

Contraintes

Contraintes capacité de mouvement

- Pas de pion dans une case, pas de capacité de mouvement :

$$\forall (i,j,t,d) \in \{1,\dots,m\}^2 \times \{1,\dots,n-1\} \times \{1,\dots,4\} \quad y_{i,j,t,d} \leq x_{i,j,t}.$$

- Capacité de mouvement vers le nord :

$$\forall (i,j,t) \in \{2,\dots,m\} \times \{1,\dots,m\} \times \{1,\dots,n-1\} \quad y_{i,j,t,1} \leq x_{i-1,j,t}$$

$$\forall (i,j,t) \in \{3,\dots,m\} \times \{1,\dots,m\} \times \{1,\dots,n-1\} \quad y_{i,j,t,1} \leq 1 - x_{i-2,j,t}.$$

$\Longrightarrow$ idem $d = 2, 3, 4.$

Contraintes

Contraintes disparition d'un pion entre chaque étape

- $\forall (i,j,t) \in [\![3, m-2]\!]^2 \times [\![1, n-1]\!]$

$$x_{i,j,t} - x_{i,j,t+1} = \sum_{d=1}^{4} y_{i,j,t,d} + y_{i+1,j,t,1} - y_{i+2,j,t,1} + y_{i-1,j,t,2} - y_{i-2,j,t,2} + y_{i,j+1,t,3} - y_{i,j+2,t,3} + y_{i,j-1,t,4} - y_{i,j-2,t,4}. \quad (2)$$

- $\forall t \in [\![1, n]\!]$

$$\sum_{i=1}^{m} \sum_{j=1}^{m} \sum_{d=1}^{4} y_{i,j,t,d} \leq 1.$$

Contraintes

$\Longrightarrow$ Problème du saut impossible en dehors de la grille

$\Longrightarrow$ Pour y répondre on considère des grilles de taille $m = \text{size}(G, 1) + 4$.

Contraintes sur les bords

- Les cases en dehors de la grille contiennent toujours des pions qui ne peuvent se mouvoir

$$\forall (i,j) \in \mathcal{C} \quad \forall t \in [\![1, n]\!] \quad x_{i,j,t} = 1 \quad ,$$

$$\forall (i,j) \in \mathcal{C} \quad \forall t \in [\![1, n]\!] \quad \forall d \in [\![1, 4]\!] \quad y_{i,j,t,d} = 0 \quad .$$

Contraintes

Contraintes début de partie

Soit $(i, j) \in [\![3, m-2]\!]^2$ (i.e. à l'intérieur de la grille)

- $G[i-2, j-2] = 0 \Longrightarrow x_{i,j,1} = 0$;
- $G[i-2, j-2] = 1 \Longrightarrow x_{i,j,1} = 1$;
- $G[i-2, j-2] = 2 \Longrightarrow$

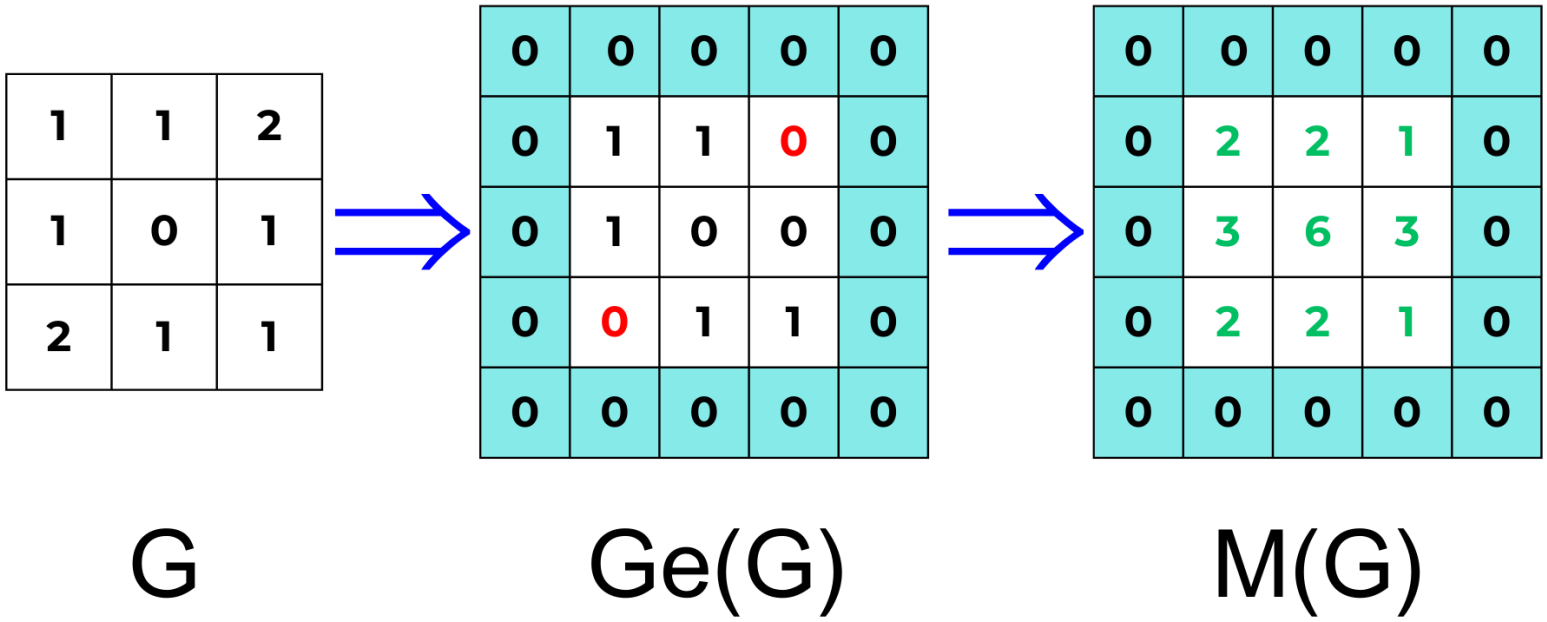
$$\begin{cases} \forall t \in [\![1, n]\!] \quad \forall d \in [\![1, 4]\!] \quad y_{i,j,t,d} = 0 \\ \forall t \in [\![1, n]\!] \quad x_{i,j,t} = 1 \end{cases}.$$

Heuristique : présentation de la méthode

4 méthodes heuristiques construitent sur un même schéma :

1. Générer à chaque étape la liste des coups admissibles de la forme $[i, j, d]$ avec $d \in \{\text{"Nord"}, \text{"Sud"}, \text{"Est"}, \text{"Ouest"}\}$ avec la fonction `List_of_possible_move`
2. Effectuer le choix du coup admissible selon l'heuristique
3. Jouer ce coup

Heuristique 1 : Méthode de l'agglomération

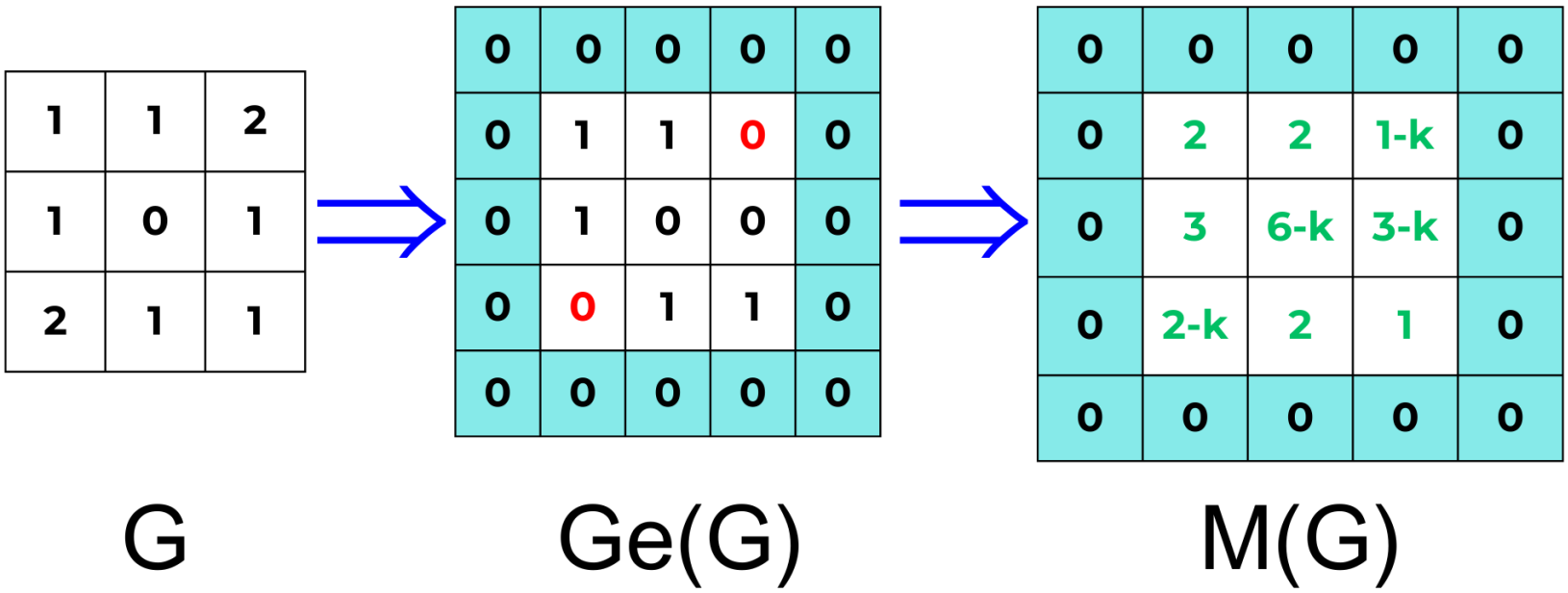


Algorithme 1 : `index_maximizing_agglomeration`

Entrée : Une grille G, une liste de coups admissibles L

Sortie : $\underset{i \in \{1,\ldots,\text{length}(L)\}}{\text{argmax}} \{\text{somme des éléments de } M(\texttt{Move}(G, L[i]))\}$

Heuristique 2 : Méthode de l'agglomération pénalisée



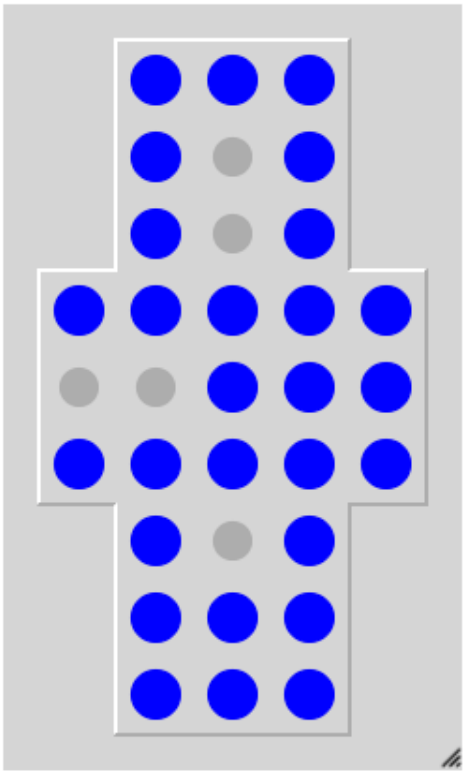
Algorithme 2 : `index_maximizing_agglomeration_wp`

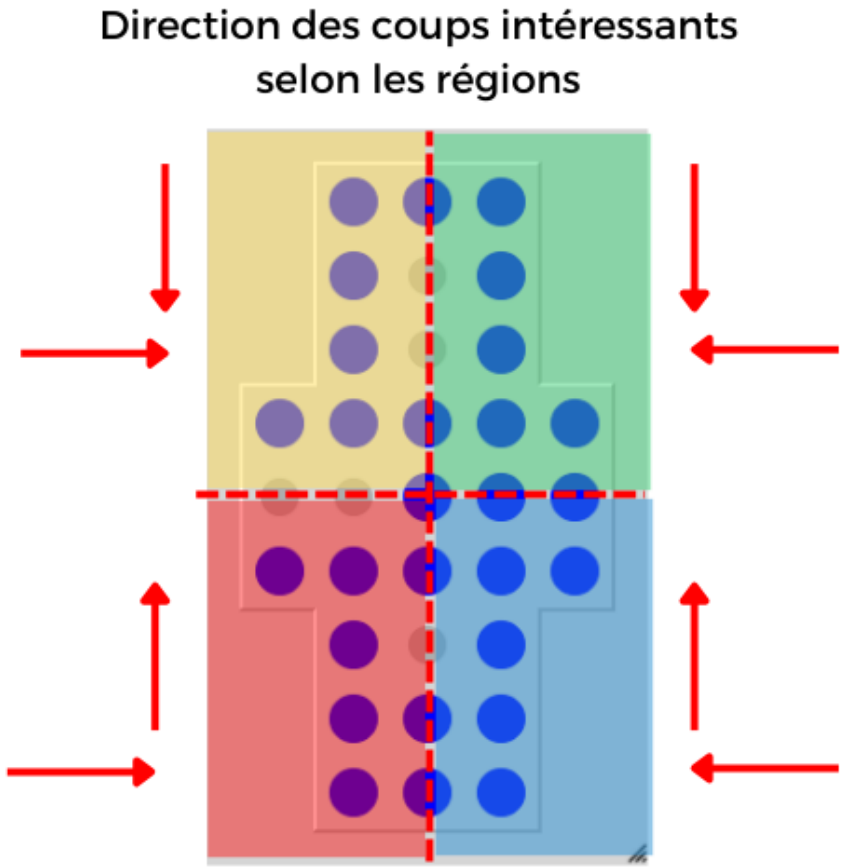
Entrée : Une grille G, une liste de coups admissibles L

Sortie : $\underset{i \in \{1,\ldots,\text{length}(L)\}}{\text{argmax}} \{\text{somme des éléments de } M(\texttt{Move}(G, L[i]))\}$

Heuristique 3 : Méthode *closer to center*

Illustration de la méthode





Génération des instances

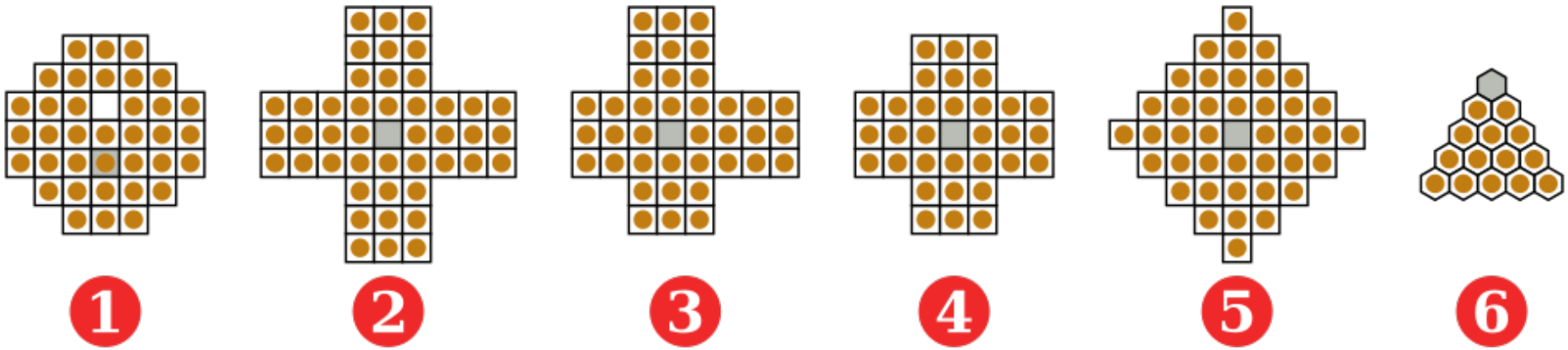


Figure 5 – Divers plateaux de jeux (d'après WIKIPÉDIA)

Génération des instances

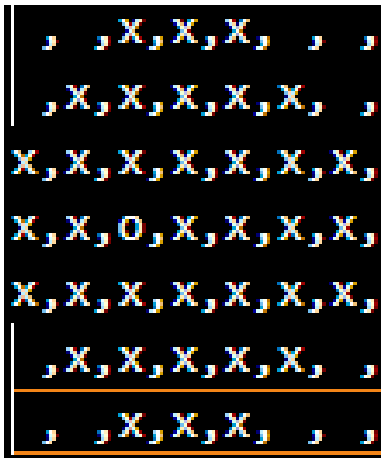


Figure 6 – Exemple de fichier de stockage d'une instance

```
   -------------
|  2  2  1  1  1  2  2  |
|  2  2  1  1  1  2  2  |
|  1  1  1  1  1  1  1  |
|  1  1  1  0  1  1  1  |
|  1  1  1  1  1  1  1  |
|  2  2  1  1  1  2  2  |
|  2  2  1  1  1  2  2  |
   -------------
```

- **Étape 1** : Choisir un format de grille (taille et type de grille).
- **Étape 2** : Créer une matrice de 0, 1 et 2 en conséquence.

Résultats

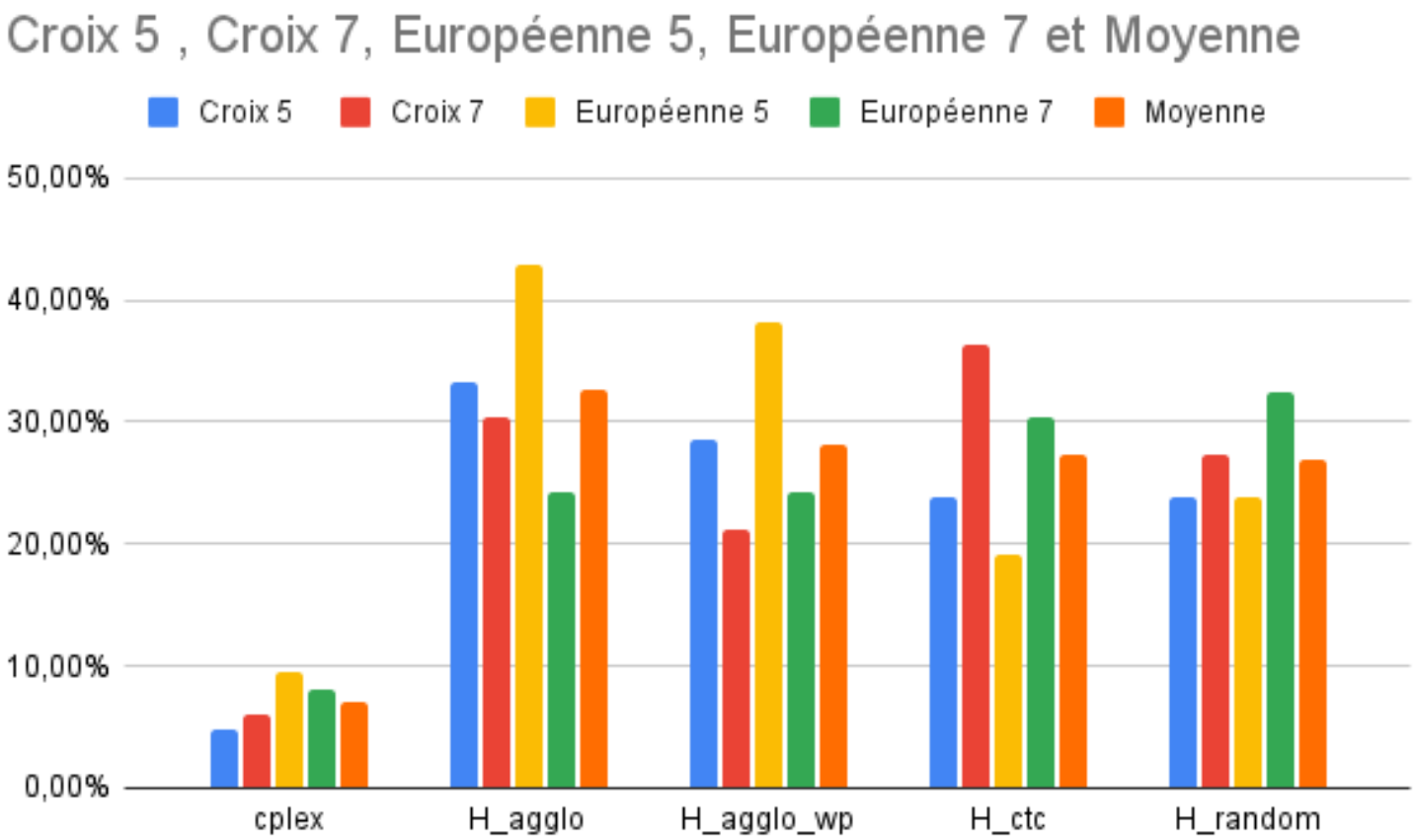


Figure 7 – Pourcentage de pions restants sur la plateau à la dernière étape en fonction des méthodes, et moyenne

Bilan Pegs

Complexité : résolution d'une grille

- Complexité temporelle : *Non polynomiale*

Points clés :

- Génération et affichage similaires voire plus simples
- cplex et heuristique invalides
- Génération de grille aléatoire très complexe

Bilan du projet

Points clés :

- cplex : contraintes et complexité
- Génération (aléatoire) et affichage
- Heuristiques
- Julia et JuMP