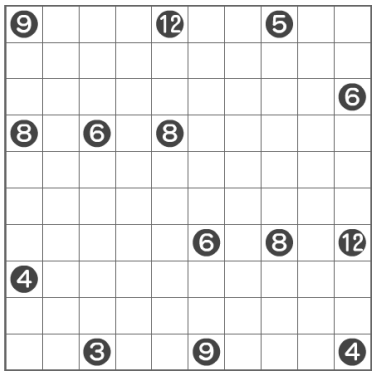
Résolution des jeux Shikaku et Hashiwokakero par programmation linéaire en nombre entiers

Andry Rafaralahy

1 Shikaku

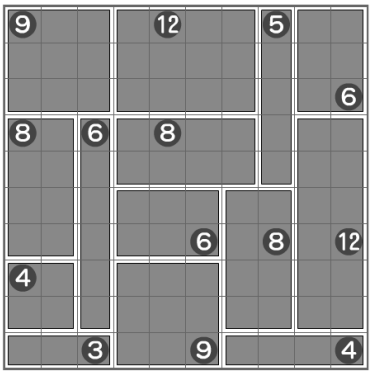
1.1 Présentation du jeu

Le jeu « Shikaku » est un jeu de logique publié par le magazine japonais Nikoli. Le but du jeu est de couvrir une grille par des rectangles. Des nombres sont inscrits dans certaines cellules de la grille. Chaque rectangle doit contenir un nombre, qui détermine le nombre de cellules qu'il couvre.



Source : Wikipédia

FIGURE 1 – Exemple de grille



Source : Wikipédia

FIGURE 2 – Exemple de grille résolue

1.2 Résolution par programmation linéaire en nombres entiers

Soit une grille de M lignes et N colonnes avec K nombres inscrits dans des cellules différentes. Le k-ième nombre a pour valeur v_k et est inscrit en position (i_k, j_k), où i_k est l'indice de ligne et j_k est l'indice de colonne. Pour tout k, notons $R_k = \{R_{k,1}, R_{k,2}, \ldots\}$ la liste des rectangles d'aire v_k, à coordonnées entières et contenant la cellule (i_k, j_k). À chaque $R_{k,l}$ on associe un masque

binaire de taille $M \times N$, c'est-à-dire une matrice de coefficients $R_{k,l,i,j}$ tels que

$$R_{k,l,i,j} = \begin{cases} 1 \text{ si } (i,j) \text{ appartient au rectangle } R_{k,l} \\ 0 \text{ sinon.} \end{cases}$$

On peut définir les variables du PLNE comme suit :

$$x_{k,l} = \begin{cases} 1 \text{ si l'on retient le rectangle } R_{k,l} \\ 0 \text{ sinon} \end{cases} \qquad \forall 1 \leq k \leq K, 1 \leq l \leq |R_k|$$

Pour modéliser le fait que chaque nombre doive avoir exactement un rectangle associé, on a la contrainte :

$$\sum_{l=1}^{|R_k|} x_{k,l} = 1 \qquad \forall 1 \leq k \leq K \tag{1}$$

La contrainte suivante traduit quant à elle le fait que les rectangles sélectionnés ne doivent pas se chevaucher.

$$\sum_{k=1}^{K} \sum_{l=1}^{|R_k|} R_{k,l,i,j} x_{k,l} = 1 \qquad \forall 1 \leq i \leq M, 1 \leq j \leq N \tag{2}$$

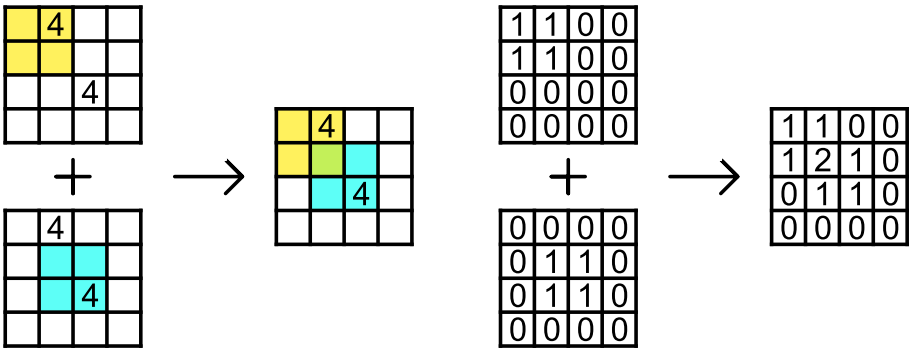


FIGURE 3 – Illustration la contrainte (2)

1. Les cases en jaune et en bleu somment à 1
2. La case en vert somme à 2 : contrainte (2) non respectée

On a donc $MN+K$ contraintes. Évaluons le nombre de variables du PLNE. Pour tout $1 \leq k \leq K$ on sait que v_k a moins de v_k diviseurs et que $v_k \leq \max(M,N)$. Pour chaque diviseur, on considère au plus v_k rectangles. Donc pour chaque k on considère au plus v_k^2 rectangles. On peut donc majorer le nombre de variables par $K \max(M,N)^2$.

1.3 Résolution heuristique

On peut résoudre le problème de manière heuristique par la méthode du recuit simulé. On appelle état la variable à optimiser, et énergie la fonction objectif. On introduit un paramètre T, qu'on appelle la température. La méthode du recuit simulé se déroule selon les étapes suivantes :

1. On part d'un état initial s, et on calcule son énergie E
2. On considère ses états voisins, et on en choisit un au hasard s_k
3. On calcule l'énergie E_k de l'état s_k, si $E_k < E$ alors on garde s_k, sinon on le garde avec une probabilité dépendant de la température T, et de l'écart $|E_k - E|$
4. on met à jour la température T
5. Si le critère d'arrêt est vérifié on s'arrête, sinon on repart de l'étape 1.

On peut choisir de s'arrêter si E passe sous un certain seuil, ou si l'on dépasse un nombre maximum d'itérations fixé. En général, on fait diminuer T au fil des itérations, par exemple en le multipliant par une constante $\lambda < 1$. Plus T est petit, plus la probabilité d'accepter un état s_k tel que $E_k \geq E$ doit être faible. Le choix des paramètres tels que la température initiale et le coefficient λ se fait de manière empirique. Enfin dans la pratique on stocke en mémoire s_{opt}, correspondant à l'état ayant réalisé l'énergie la plus basse au fil des itérations. Ci-dessous le pseudo-code de l'algorithme :

Résultat : État optimal s_{opt}
$s := s_0$;
$E :=$ energy(s);
$s_{opt} := s$;
$E_{opt} := E$;
$k := 1$;
tant que $k \leq k_{max}$ *et* $E > E_{min}$ **faire**
  $s_k :=$ voisin(s);
  $E_k :=$ energy(s_k);
  si $E_k < E$ *ou* rand() $< P(E_k - E, T)$ **alors**
    $E := E_k$;
    $s := s_k$;
    si $E < E_{opt}$ **alors**
      $E_{opt} := E$;
      $s_{opt} := s$;
    fin
  fin
fin
retourner s_{opt}

Algorithme 1 : Algorithme du recuit simulé

Pour notre jeu, on représente un état par un K-uplet $(l_1, \ldots, l_k)$ qui correspond au choix de K rectangles R_{l_k}. Les états voisins de $(l_1, \ldots, l_k)$ est sont les K-uplets $(l'_1, \ldots, l'_K)$ tels qu'il existe k_0 tel que $l_{k_0} \neq l'_{k_0}$ et $l_k = l'_k$ pour tout

$k \neq k_0$. La fonction objectif à minimiser est :

$$f(l_1, \ldots, l_k) = \sum_{i=1}^{M} \sum_{j=1}^{N} \left(\sum_{k=1}^{K} \mathbf{1}((i,j) \in R_{l_k}) \right)^2 \tag{3}$$

Ici $(i,j) \in R_{l_k}$ est à comprendre au sens géométrique, c'est-à-dire que $(i,j) \in R_{l_k}$ si et seulement si la cellule de coordonnées (i,j) est comprise dans le rectangle R_{l_k}. Minimiser f revient à minimiser le nombre de chevauchements. On remarque qu'un état $(l_1, \ldots, l_k)$ est solution du jeu si et seulement si $f(l_1, \ldots, l_k) = MN$.

1.4 Génération d'instances de jeu

Pour générer une instance de jeu, l'approche la plus simple consiste à partir d'un rectangle de taille $M \times N$ et de le découper récursivement. Plus précisément, à chaque itération on choisit au hasard si l'on fait une découpe horizontale ou verticale, en ayant une plus grande probabilité de faire une découpe dans la dimension où le rectangle est le plus grand. On répète le processus sur les deux parties ainsi obtenues. On s'arrête lorsqu'on ne peut plus couper ou que l'on a atteint un nombre maximal d'itérations fixé au préalable.

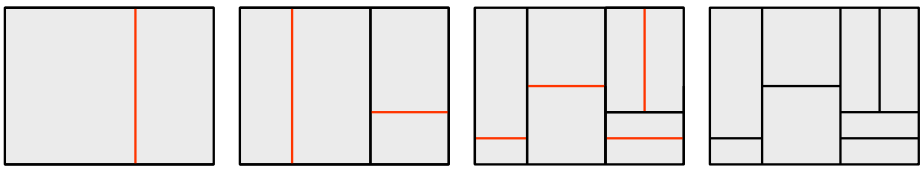


FIGURE 4 – Découpe d'un rectangle en 3 itérations

L'inconvénient de cette approche est que certaines configurations de jeu ne sont pas réalisables. Voici un exemple de découpe qui n'est pas possible par cette méthode :

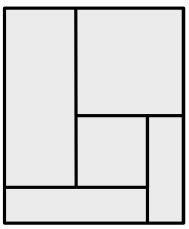


FIGURE 5 – Configuration impossible à obtenir par l'approche « naïve »

Heureusement, on peut facilement pallier ce problème en fusionnant des rectangles contigus de manière aléatoire, juste après avoir effectué la découpe.

Ensuite, il suffit de placer un nombre par rectangle, correspondant à son aire. Les nombres sont placés aléatoirement dans leur rectangle. La fonction `generateInstance` utilisée pour générer des instances a pour paramètres :

- `m`, le nombre de lignes
- `n`, le nombre de colonnes
- `depth`, la profondeur de récursion. Plus elle est élevée, plus il y a de rectangles.
- `nbMergeIter`, le nombre maximum d'itérations de fusion. Plus il est élevé, plus les configurations comme celle de la figure. 1.4 sont favorisées, mais aussi moins il y a de rectangles.
- `mergeProb`, la probabilité de fusionner deux rectangles compatibles

1.5 Comparaison des méthodes de résolution

On compare trois méthodes de résolution : résolution du PLNE avec `Gurobi`, résolution du PLNE avec `Cbc`, et résolution par recuit simulé.

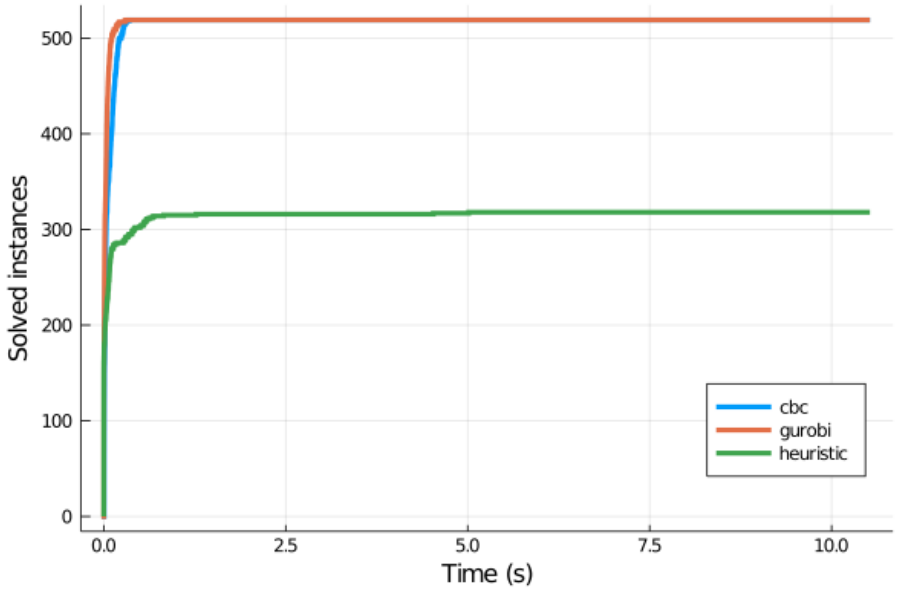


FIGURE 6 – Diagramme de performance, pour $M = N = 4, 9, 16, 25$

La figure ci-dessus a été obtenue en testant la résolution de grilles carrées de tailles 4×4, 9×9, 16×16 et 25×25. Pour chaque taille de grille N, on a fait varier le paramètre `depth` entre 1 et $\lfloor N/2 \rfloor$, et `nbMergeIter` entre 0 et 3. On y observe que la résolution par `Gurobi` est un peu plus rapide que celle par `Cbc` qui elle même plus rapide que la résolution par recuit simulé. On voit également que la résolution par recuit simulé ne donne pas toujours une solution optimale.

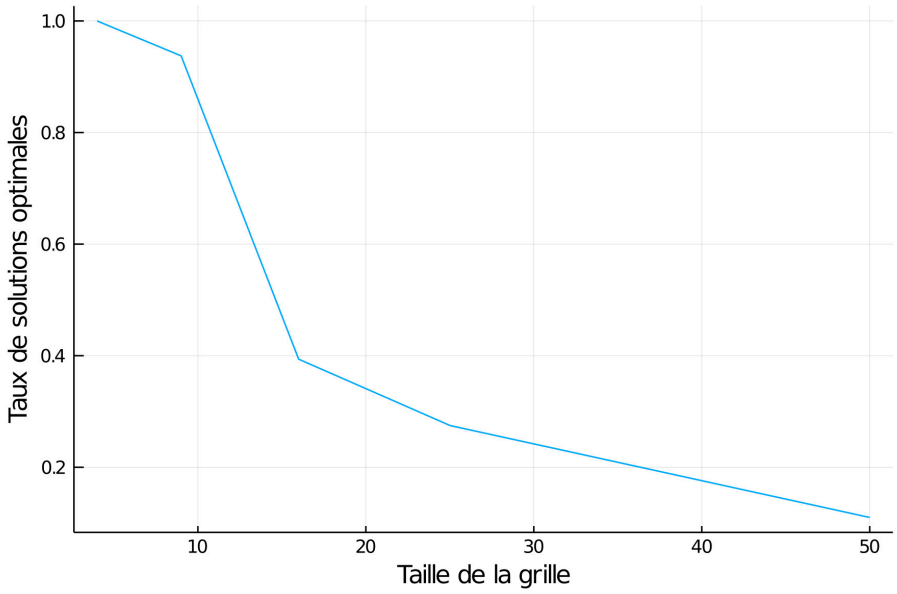


FIGURE 7 – Taux de solution optimale pour le recuit simulé

Pour obtenir la figure ci-dessus, le recuit simulé a été testé sur des grilles avec les mêmes paramètres que précédemment mais en ajoutant des grilles de tailles 50×50. Le taux de solution optimale pour le recuit simulé est assez élevé pour des petites grilles (1 pour les grilles de taille 4, $0,95$ pour les grilles de tailles 9), mais décroît très rapidement quand on augmente la taille de la grille.

2 Hashiwokakero

2.1 Présentation du jeu

Le jeu « Hashiwokakero » est aussi issu du magazine japonais Nikoli. Le but du jeu est de relier les îlots par des ponts de façon à former un graphe connexe. Les ponts doivent obéir à certaines règles :

- Ils ne peuvent relier que des îlots étant sur le même axe horizontal ou vertical
- Ils ne peuvent pas se croiser, et ne peuvent pas traverser un îlot
- Deux îlots sont reliés par au plus deux ponts
- Le nombre inscrit sur chaque îlot doit être égal au nombre de ponts connectés à celui-ci

FIGURE 8 – Exemple de jeu

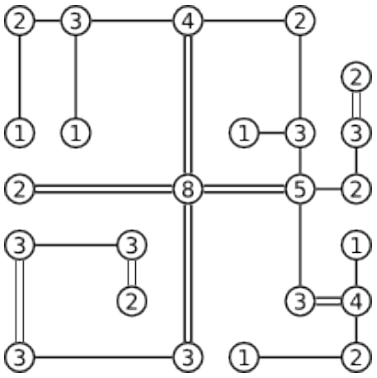


FIGURE 9 – Exemple de jeu résolu

2.2 Résolution par programmation linéaire en nombres entiers

Soit $V = \{(i_1, j_1), \ldots, (i_N, j_N)\}$ un ensemble de points à coordonnées entières et $v \colon V \to \mathbb{N}^*$ leur valuation. Par commodité on notera $v_k := v(i_k, j_k)$. Les variables du PLNE sont :

$$x_{k,l} = \begin{cases} 1 \text{ si l'on retient l'arête entre } (i_k, j_k) \text{ et } (i_l, j_l) \\ 0 \text{ sinon} \end{cases} \qquad \forall 1 \leq k \leq N, 1 \leq l \leq k$$

et

$$y_{k,l} = \begin{cases} 1 \text{ si l'arête entre } (i_k, j_l) \text{ et } (i_k, j_l) \text{ est simple} \\ 2 \text{ si l'arête entre } (i_k, j_l) \text{ et } (i_k, j_l) \text{ est double} \\ 0 \text{ s'il n'y a pas d'arête} \end{cases} \qquad \forall 1 \leq k \leq N, 1 \leq l \leq k$$

Pour qu'il n'y ait que des arêtes entre sommets qui soient sur la même ligne ou la même colonne on utilise la contrainte

$$x_{k,l} = 0 \qquad \forall k, l \text{ tels que } i_k \neq i_l \text{ et } j_k \neq j_l \tag{4}$$

Pour qu'un sommet ne puisse pas être relié à lui-même

$$x_{k,k} = 0 \qquad \forall 1 \leq k \leq N \tag{5}$$

Pour tous k_1, l_1, k_2, l_2 tels que les segments $(i_{k_1}, j_{k_1}) - (i_{l_1}, j_{l_1})$ et $(i_{k_2}, j_{k_2}) - (j_{k_2}, j_{l_2})$ on introduit la contrainte suivante

$$x_{k_1,l_1} + x_{k_2,l_2} \leq 1 \tag{6}$$

On remarque au passage que les variables en x sont très utiles pour modéliser les contraintes d'intersection. Il aurait été plus difficile de les prendre en compte avec seulement les variables en y.
Pour que les ponts soient bien simples ou doubles, et assurer la cohérence entre x et y

$$x_{k,l} \leq y_{k,l} \leq 2x_{k,l} \qquad \forall 1 \leq k \leq N, 1 \leq l \leq k \tag{7}$$

Enfin, pour que la valuation des sommets soit bien respectée

$$\sum_{l=1}^{N} y_{k,l} = v_k \qquad \forall 1 \leq k \leq N \tag{8}$$

Il est difficile de formuler une contrainte linéaire pour exprimer la connexité. On choisit donc d'utiliser la fonction objectif pour cela. On cherche à maximiser le nombre d'arêtes, sans compter leur multiplicité. La fonction objectif que l'on utilise est :

$$\sum_{k=1}^{N} \sum_{l=1}^{N} x_{k,l} \tag{9}$$

On vérifie expérimentalement qu'ajouter ce critère de maximisation favorise la connexité des solutions :

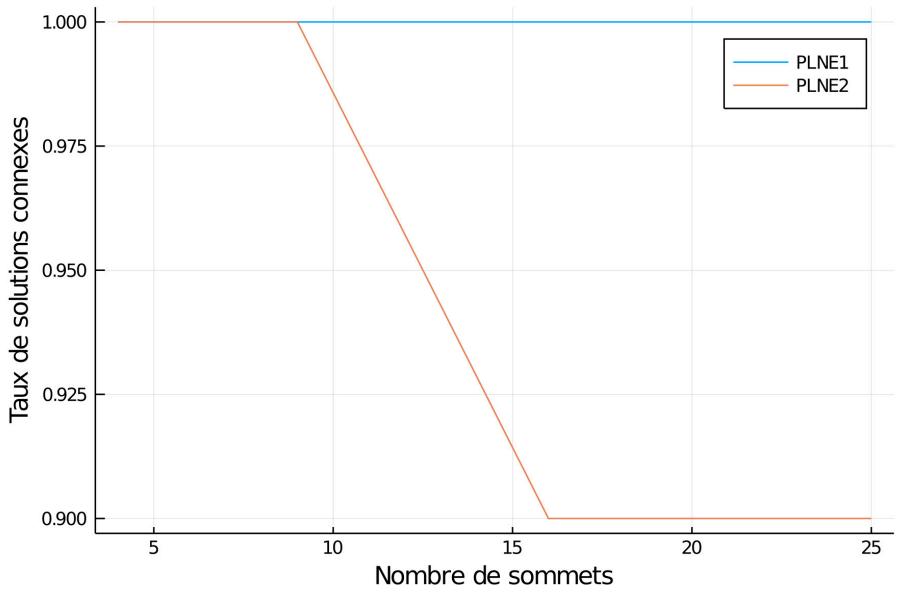


FIGURE 10 – Comparaison du taux de solutions connexes

Sur la figure ci-dessus, la courbe bleue (respectivement rouge) représente le taux de solutions connexes obtenues par résolution du PLNE avec le critère de

maximisation (9) (respectivement sans le critère). Sur tous les tests menés, il n'y a eu aucun contre-exemple de solution non connexe trouvée avec ce critère. A contrario, on observe que le taux de solutions connexes diminue quand le nombre d'îlots augmente lorsque l'on n'utilise pas ce critère.
Ce PLNE a $N(N+1)$ variables. On a N contraintes de type (5), moins de N^2 contraintes de type (4), moins de N^4 contraintes de type (6), $N(N+1)/2$ contraintes de type (7) et N contraintes de type (8). On peut donc majorer le nombre de contraintes par $(N+1)^4$.

2.3 Résolution heuristique

Comme pour le jeu des rectangles, on peut utiliser la méthode du recuit simulé (voir 1.3). Pour ce jeu, on représente un état par un ensemble d'arêtes $E = \{(c_1, \{x_1, y_1\}), \ldots, (c_{|E|}, \{x_{|E|}, y_{|E|}\})\}$. Les c_i sont les valuations des arêtes et les x_i et y_i en sont les extrêmités. c_i vaut 1 pour un pont simple, 2 pour un pont double. Pour passer à un état voisin on effectue l'une des actions suivantes :

- ajouter une arête
- retirer une arête
- modifier la valuation d'une arête

La probabilité de chaque action dépend du nombre d'arêtes de l'état. Par exemple, s'il y a peu d'arêtes, la probabilité d'ajouter une arête est haute. La fonction objectif à minimiser est donnée par

$$f(E) = C(V,E) + \Delta W(V,E)^2 - |E| \tag{10}$$

où $C(V,E)$ est le nombre d'intersections dans le graphe (V,E), $|E|$ est le nombre d'arête et $\Delta W(V,E)$ est un terme pénalisant le non respect des valuations des sommets

$$\Delta W(V,E) = \sum_{x \in V} \left(v(x) - \left(\sum_{(c,\{x,y\}) \in E} c \right) \right)^2$$

Le terme $-|E|$ sert à favoriser la connexité de la solution obtenue, comme pour le PLNE.

2.4 Génération d'instances de jeu

Pour générer une instance jeu on procède selon les étapes suivantes :

1. on génère un ensemble de sommets aléatoires
2. on génère toutes les arêtes possibles pour ces sommets, sans se soucier des intersections
3. on défait les éventuelles intersections

Pour défaire les intersections, on rajoute un sommet partout là où les arêtes s'intersectent.

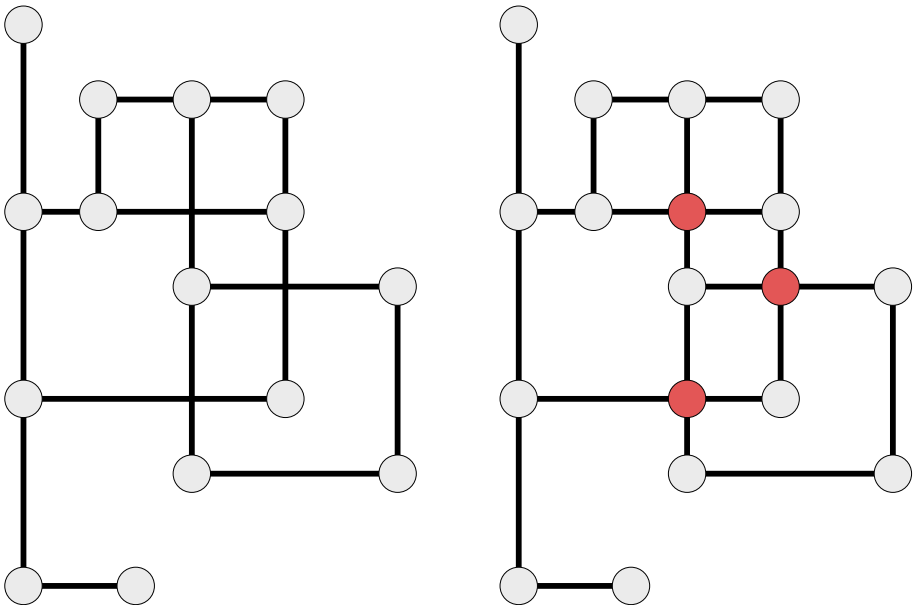


FIGURE 11 – Résolution des intersections

Ensuite il n'y a plus qu'à attribuer un poids au hasard (1 pour un pont simple, 2 pour un pont double) à chaque arête, et à calculer les valeurs de chaque sommet. La fonction de génération d'instance prend en paramètre un entier `n` qui correspond au nombre de sommets initial. Le graphe final contient évidemment plus de sommets que le graphe initial car on ajoute des sommets pour résoudre les intersections.

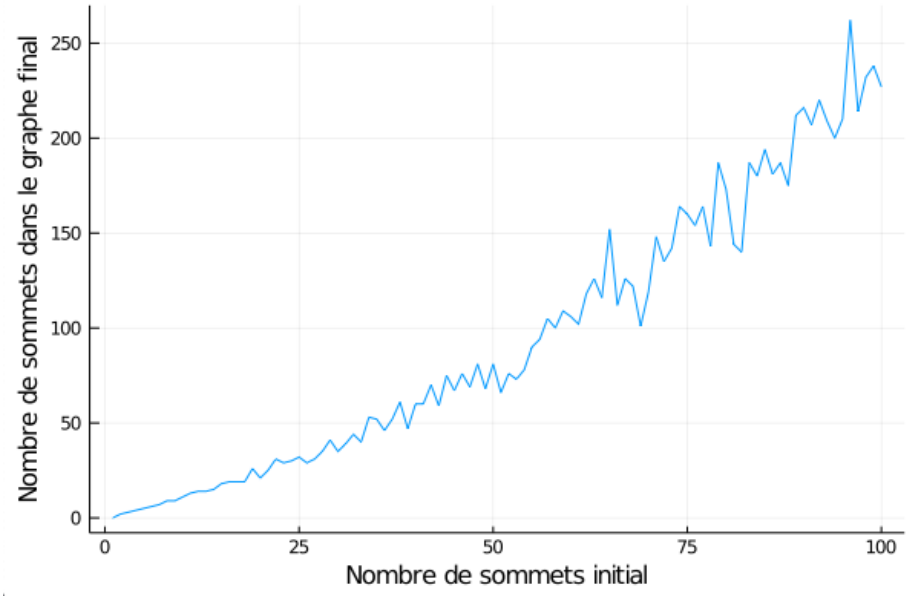


FIGURE 12 – Nombre de sommets en fonction du paramètre `n`

2.5 Comparaison des méthodes de résolution

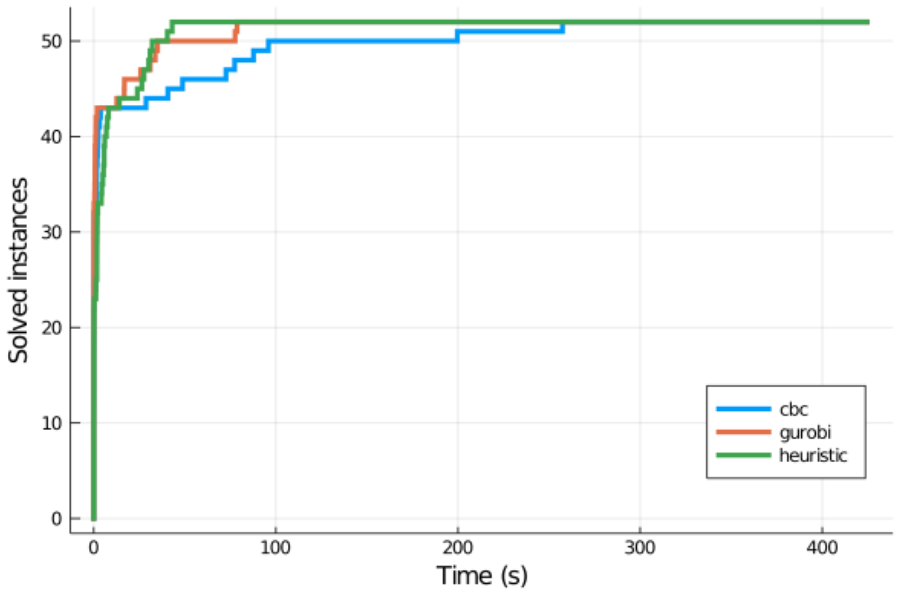


FIGURE 13 – Diagramme de performance

La figure ci-dessus a été obtenue en testant la résolution de jeux avec un nombre d'îlots allant de 4 à 80. Encore une fois la résolution avec `Gurobi` est plus rapide que celle avec `Cbc`. Contrairement à ce qu'on avait pour le jeu des rectangles, la méthode du recuit simulé donne une solution valide pour tous les exemples testés et est plus rapide que la résolution par `Gurobi` et `Cbc`.

Références

WIKIPEDIA : Recuit simulé — Wikipedia, the free encyclopedia. `http://fr.wikipedia.org/w/index.php?title=Recuit%20simul%C3%A9&oldid=158165306`, 2020. [Online ; accessed 30-April-2020].