MOGPL : La balade du robot

Etudiant : **Nils Barrellon 21401602**

Github associé : https://github.com/nbarrellon/RITAL2026

grilleexemple.png	grilleexemple2.png	grilleexemple3.png

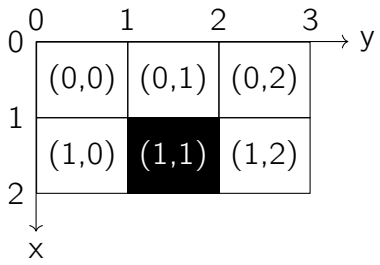
a. Formulation du problème

En première lecture, on se dit que, le robot passant d'une intersection de rail à une intersection de rail, le problème peut aisément se modéliser à l'aide d'un graphe où les sommets sont les coordonnées des intersections. On imagine supprimer du graphe les sommets qui sont interdits par la présence d'obstacle. Mais c'est oublier que le robot peut tourner quand il est sur une intersection et que cette opération est de valeur 1 (tout comme un déplacement). Il convient donc de créer, pour des mêmes coordonnées, autant de sommets qu'il y a d'orientation possibles pour le robot. Ainsi, passer d'une orientation à une autre (c'est à dire tourner) revient à passer d'un sommet à un autre.

J'ai donc opté pour l'implémentation suivante :

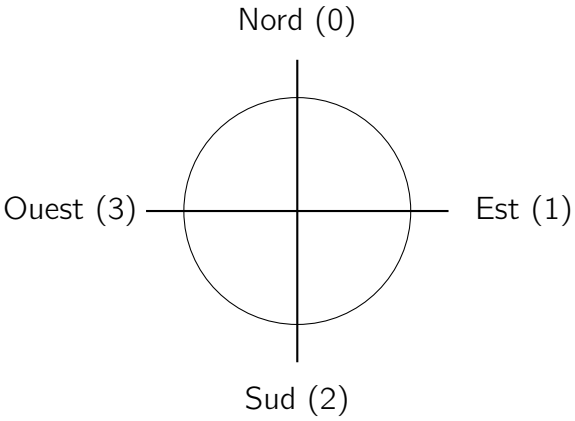
- la grille est modélisée par un graphe orienté G(V,E) ;
- chaque sommet V est un état du robot caractérisé par des coordonnées sur la grille qui lui sont accessibles et une orientation (Nord,Sud,Est,Ouest) ;
- Un arc traduit la possibilité pour le robot de passer d'une intersection à une autre (AVANCE) ou de tourner sur lui-même (TOURNE).

Soit la grille 2×3 suivante :

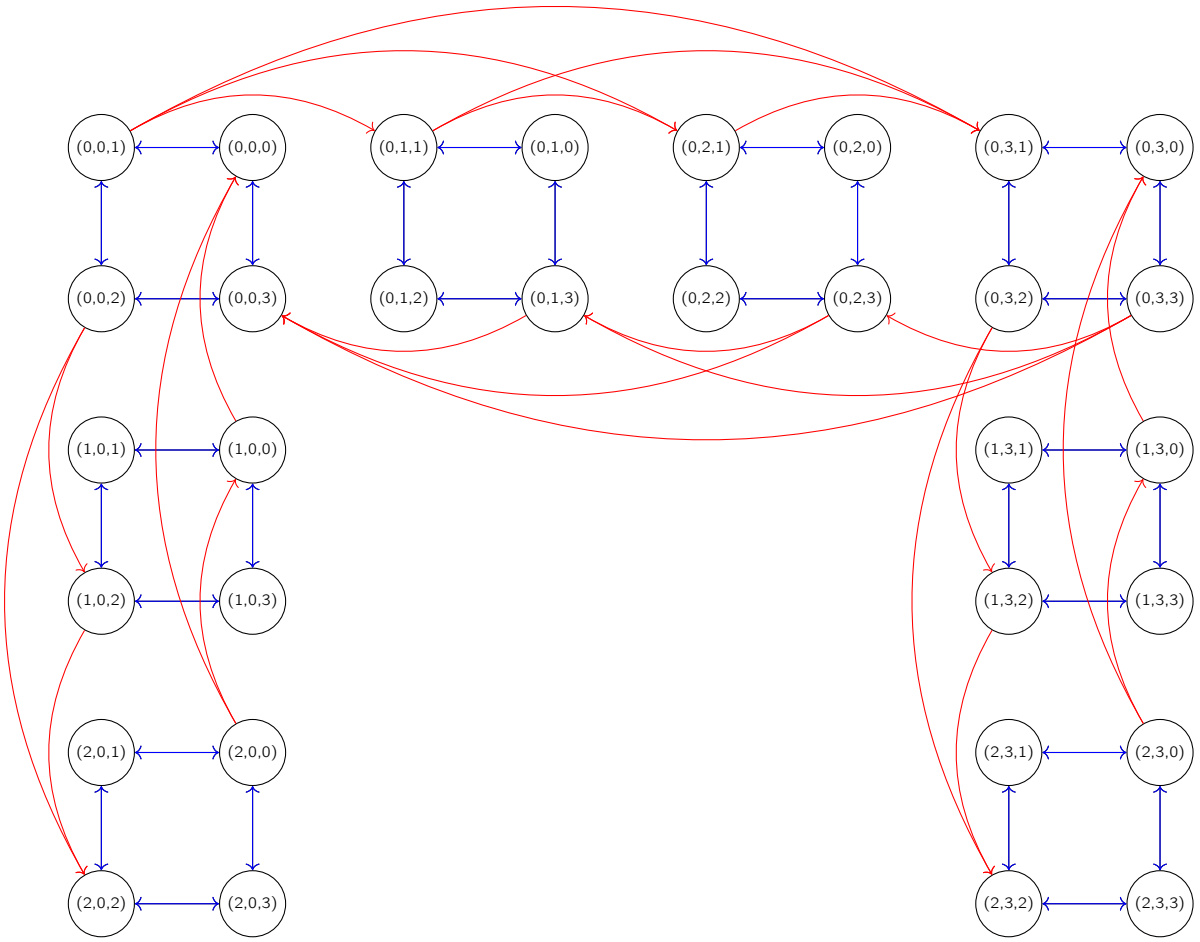


Les sommets (1,1), (1,2), (2,1) et (2,2) ne doivent pas figurer dans le graphe car le robot n'y a pas accès.

J'ai choisi d'implémenter les orientations du robot dans la grille de la façon suivante :



Ainsi, le graphe associé est le suivant où les arcs AVANCE sont en rouge et les arcs TOURNE en bleu :



On ontient ainsi un graphe d'étude G(V,E) :

- chaque intersection de rail génère 4 sommets (correspondant aux 4 orientations possibles) d'où $|V| \leq 4 \times (NM - 4 \times P)$ où P est le nombre d'obstacle (1 obstacle supprime 4 sommets).
- chaque état possède au mieux 4 arcs (pour "tourner" d'une orientation à l'autre) et 3 arcs (pour avancer si possible vers un autre sommet) soir $|E| \leq 7(NM - 4 \times P)$.

On constate donc que la taille du graphe est en N^2.

b. Complexité de l'algorithme

Le plus court chemin dans un graphe orienté non pondéré (comme c'est ici notre cas puisque chaque opération coûte 1) peut être trouvé à l'aide d'un **parcours en largeur** (ou BFS). C'est celui que j'ai décidé d'utiliser. Sa complexité dans le pire cas est en $\mathcal{O}(n+m)$ où n est le nombre de sommet du graphe $n = |V|$ et m le nombre d'arête $m = |E|$.

c. Temps de calcul en fonction de la taille de la grille

En théorie, la complexité dans le pire des cas de mon algorithme est en $\mathcal{O}(|V|+|E|)$. Toutefois, nous avons montré que l'ordre du graphe et le nombre d'arête varient en N^2 ainsi, je m'attends à une complexité du parcours en $\mathcal{O}(N^2)$.

Les résultats expérimentaux obtenus pour des matrices carrées ($N \leq 60$) sont présentées figures 1 et 2.

Pour chaque $N \in \{10, 70\}$, 20 instances de graphes sont générées avec P obstacles ($P = N$) pour obtenir un temps d'exécution moyen de l'algorithme BFS. J'ai choisi de positionner pour chacun des tests le départ en (0,0) et l'arrivée en (N,N) car, en tirant au hasard ces positions extrêmes on peut obtenir des parcours très courts (départ très voisin d'arrivée) et donc un temps d'exécution non représentatif. Malgré tout, des mesures surprenantes apparaissent (voir figure 1 pour N = 60).

instanceXXX_log.png

Figure 1 : Temps d'éxécution en fonction de la taille de la grille pour un nombre d'obstacle constant.

instanceXXX_log1.png

Figure 2 : Temps d'éxécution en fonction de la taille de la grille pour un nombre d'obstacle constant.

La courbe $t = f(N)$ a l'allure d'une courbe polynomiale de forme générale $t = f(n) = n^\alpha$. Pour déterminer le degré du polynome, je trace la courbe $log(t) = g(log(n))$. En effet, si $f(n) = n^\alpha$ alors $log(t) = log(n^\alpha) = \alpha \times log(n)$.
Si l'hypothèse est correcte alors cette courbe est une droite dont le coefficient directeur vaut α. L'hypothèse est vérifiée expérimentalement puis que j'obtiens, après régression linéaire, $\alpha \approx 2$ soit une complexité temporelle dans le pire des cas $\boxed{\mathcal{O}(n^2)}$ conforme à ce que attendu.

Remarque : Cette complexité aurait pu peut-être être légèrement améliorée en supprimant les sommets inutiles. En effet, sur les bords de la grille, la création des sommets correspondant à une orientation pointant hors de la grille sont inutiles.

d. Temps de calcul en fonction du nombre d'obstacle

La complexité en fonction du nombre d'obstacle P devrait tendre vers une complexité linéaire conforme à celle du BFS. En effet, plus le nombre d'obstacle augmente, plus l'ordre du graphe et sa densité diminue. Nous avons montré que $|V| \leq 4 \times (NM - 4 \times P) = 4(N^2 - 4P)$ soit $\lim_{P \to N^2} |V| = 0$ pour une complexité qui devient alors constante.

instance_taille_grille_cte1.png

Figure 3 : Temps d'exécution en fonction du nombre d'obstacles si la taille de la grille est constante.

instance_taille_grille_cte.png

Figure 4 : Temps d'exécution en fonction du nombre d'obstacles si la taille de la grille est constante.

La figure 3 montre comment le temps de calcul de l'algorithme tend vers 0 quand P augmente pour une grille de taille fixe. On notera que cette limite semble atteinte pour $P = 100$ ce qui

paraît correct puisque nous avons dit que P obstacles supprimaient $4P$ sommets dans le graphe. Or, notre grille étant de taille 20 par 20, elle comprend au maximum 400 sommets.

Pour un nombre d'obstacles variant de 10 à 50 dans une grille de taille 20 par 20. Les résultats expérimentaux de la figure 4 font bien apparaître une complexité linéaire.

e. Positionnement des obstacles

On souhaite écrire le programme linéaire qui va choisir P cases-obstacles dans une grille de taille $N \times M$ dont chaque case est pondérée par un entier tiré aléatoirement entre 1 et 1000. La somme des cases choisies doit être minimale et les contraintes suivantes doivent être respectées :

- il n'y ait pas plus de $\frac{2P}{M}$ obstacles par ligne $\rightarrow$ jeu de contraintes ① ;
- il n'y ait pas plus de $\frac{2P}{N}$ obstacles par colonne $\rightarrow$ jeu de contraintes ② ;
- il n'y ait aucune configuration de type "obstacle-cas vide-obstacle" sur les lignes $\rightarrow$ jeu de contraintes ③ ;
- il n'y ait aucune configuration de type "obstacle-cas vide-obstacle" sur les colonnes $\rightarrow$ jeu de contraintes ④ ;
- il y ait P obstacles dans la grille $\rightarrow$ jeu de contraintes ⑤ ;

1) Variables de décision :

J'ai décidé de choisir comme variables de décision, $x_{ij}, i \in [1, 2..., N]$ et $j \in [1, 2..., M]$ en attribuant donc une variable par case. Ces variables x_{ij} sont **binaires** : 0 la case n'est pas retenue, 1 elle est choisie.

2) Fonction objectif

On souhaite minimiser la somme des coefficients de pondération des cases choisies. Ainsi, la fonction objectif z s'écrit :

$$\min z = \sum_{i=1}^{N} \sum_{j=1}^{M} x_{i,j}$$

3) Contraintes

① $\forall i \in \{1, \ldots, N\}$ on veut $\sum_{j=1}^{M} x_{ij} \leq \frac{2P}{M}$

② $\forall j \in \{1, \ldots, M\}$ on veut $\sum_{i=1}^{N} x_{ij} \leq \frac{2P}{N}$

③ $\forall i \in \{1, \ldots, N\}$, $\forall j \in \{1, \ldots, M-2\}$ on veut $x_{i,j} + x_{i,j+2} \leq 1$ qui assure que deux cases entourant une case vide dans une ligne ne peuvent pas être choisies simultanément

④ $\forall j \in \{1, \ldots, M\}$, $\forall i \in \{1, \ldots, N-2\}$ on veut $x_{i,j} + x_{i+2,j} \leq 1$ qui assure que deux cases entourant une case vide dans une colonne ne peuvent pas être choisies simultanément

⑤ $\sum_{i=1}^{N} \sum_{j=1}^{M} x_{i,j} = P$

4) Exemple

Soit la grille de 4×4 de la figure 5.

132	12	34	789
202	345	70	707
11	236	657	90
178	801	900	654

Figure 5 : Grille de jeu avec pondération

Les variables de décision associées sont :

x_{11}	x_{12}	x_{13}	x_{14}
x_{21}	x_{22}	x_{23}	x_{24}
x_{31}	x_{32}	x_{33}	x_{34}
x_{41}	x_{42}	x_{43}	x_{44}

Figure 6 : Variables de décision

Si on souhaite placer $P = 5$ obstacles dans cette grille, les contraintes s'écrivent :
$N = 4$ contraintes ①

$$\begin{cases} x_{11} + x_{12} + x_{13} + x_{14} \leq 2 \\ x_{21} + x_{22} + x_{23} + x_{24} \leq 2 \\ x_{31} + x_{32} + x_{33} + x_{34} \leq 2 \\ x_{41} + x_{42} + x_{43} + x_{44} \leq 2 \end{cases}$$

$M = 4$ contraintes ②

$$\begin{cases} x_{11} + x_{21} + x_{31} + x_{41} \leq 2 \\ x_{12} + x_{22} + x_{32} + x_{42} \leq 2 \\ x_{13} + x_{23} + x_{33} + x_{43} \leq 2 \\ x_{14} + x_{24} + x_{34} + x_{44} \leq 2 \end{cases}$$

$N \times (M - 2) = 8$ contraintes ③

$$\begin{cases} x_{11} + x_{13} \leq 1 \\ x_{12} + x_{14} \leq 1 \\ x_{21} + x_{23} \leq 1 \\ x_{22} + x_{24} \leq 1 \\ x_{31} + x_{33} \leq 1 \\ x_{32} + x_{34} \leq 1 \\ x_{41} + x_{43} \leq 1 \\ x_{42} + x_{44} \leq 1 \end{cases}$$

$M \times (N - 2) = 8$ contraintes ④

$$\begin{cases} x_{11} + x_{31} \leq 1 \\ x_{21} + x_{41} \leq 1 \\ x_{12} + x_{32} \leq 1 \\ x_{22} + x_{42} \leq 1 \\ x_{13} + x_{33} \leq 1 \\ x_{33} + x_{43} \leq 1 \\ x_{14} + x_{34} \leq 1 \\ x_{24} + x_{44} \leq 1 \end{cases}$$

1 contrainte ⑤

$$x_{11} + x_{12} + x_{13} + x_{14} + x_{21} + x_{22} + x_{23} + x_{24} + x_{31} + x_{32} + x_{33} + x_{34} + x_{41} + x_{42} + x_{43} + x_{44} = 5$$

Sachant que l'on travaille en minimisation (ajouter une sixième variable ne pourrait que dégrader la fonction objectif), je l'ai modifiée pour Gurobi par :

$$x_{11} + x_{12} + x_{13} + x_{14} + x_{21} + x_{22} + x_{23} + x_{24} + x_{31} + x_{32} + x_{33} + x_{34} + x_{41} + x_{42} + x_{43} + x_{44} \geq 5$$

puis multipliée par -1 pour avoir une homogénéité des signes d'inégalités (toujours pour Gurobi) :

$$-x_{11} - x_{12} - x_{13} - x_{14} - x_{21} - x_{22} - x_{23} - x_{24} - x_{31} - x_{32} - x_{33} - x_{34} - x_{41} - x_{42} - x_{43} - x_{44} \leq -5$$

5) Solution

Quelques boucles `for` permettent de construire aisément les listes des coefficients de chaque contrainte que l'on ajoute à la matrice des contraintes, idem pour les seconds membres et la fonction objectif. Tout ceci est traité par Gurobi en prenant soin d'"écraser" la matrice des variables et de lui préciser que ces dernières sont binaires (`vtype=GRB.BINARY`). Gurobi renvoie la solution :

$$x_1 = 0,\ x_2 = 1,\ x_3 = 1,\ x_4 = 0,\ x_5 = 0,\ x_6 = 0,\ x_7 = 1,\ x_8 = 0$$
$$x_9 = 1,\ x_{10} = 0,\ x_{11} = 0,\ x_{12} = 1,\ x_{13} = 0,\ x_{14} = 0,\ x_{15} = 0,\ x_{16} = 0$$

Qui correspond à la grille de la figure 7 de poids total $p = 12 + 34 + 70 + 11 + 90 = 217$:

0	1	1	0
0	0	1	0
1	0	0	1
0	0	0	0

Figure 7 : Solution du problème

6) Remarques

Les contraintes imposées génèrent une contrainte sur le nombre maximal d'obstacle que l'on peut placer dans une grille de taille $N \times M$.
Ainsi, le solveur annonce "`Model is infeasible`" pour certaines configurations (N,M,P).
Une de ces contraintes est aisément exprimable : on ne peut pas doit avoir plus de $\frac{2P}{M}$ obstacles sur une ligne ainsi il faut que $\frac{2P}{M} \leq M \Leftrightarrow P \leq \frac{M^2}{2}$. Par le même raisonnement sur les colonnes, on a $P \leq \frac{N^2}{2}$. Ainsi, on trouve une condition sur P : $P \leq \frac{NM}{2}$
J'ai intégré cette condition dans le programme Python, en restreignant les choix possibles de

l'utilisateur à N inférieur à 50% du nombre de cases de la grille. Cela fonctionne très bien pour les matrices carrées.
Néanmoins, la contrainte du "101" fait chuter cette borne supérieure dans le cas des grilles rectangulaires. Hélas, je n'ai pas réussi à expliciter cette limite.
Expérimentalement, Gurobi annonce que le problème est infaisable pour une grille de taille 7×4 à partir de 5 obstacles(on voit sur la figure 8 que l'ajout d'un cinquième obstacle dans la grille n'est pas possible sans violer une contrainte sachant qu'on ne peut pas mettre plus de $\frac{2*4}{4} = 2$ obstacle sur une ligne et $\frac{2*4}{7} \approx 1$ sur une colonne), à partir de 15 obstacles pour une grille de 15×8.

0	0	1	0
0	1	0	0
0	0	0	0
0	0	0	1
0	0	0	0
0	0	0	0
1	0	0	0

Figure 8 : Solution du problème

J'ai donc récupéré le statut de la modélisation pour :

- s'il vaut `GRB.OPTIMAL`, j'affiche la grille et le chemin trouvé ;
- s'il vaut `GRB.INFEASIBLE`, j'affiche un message d'erreur demandant à l'utilisateur de diminuer le nombre d'obstacles dans la grille.