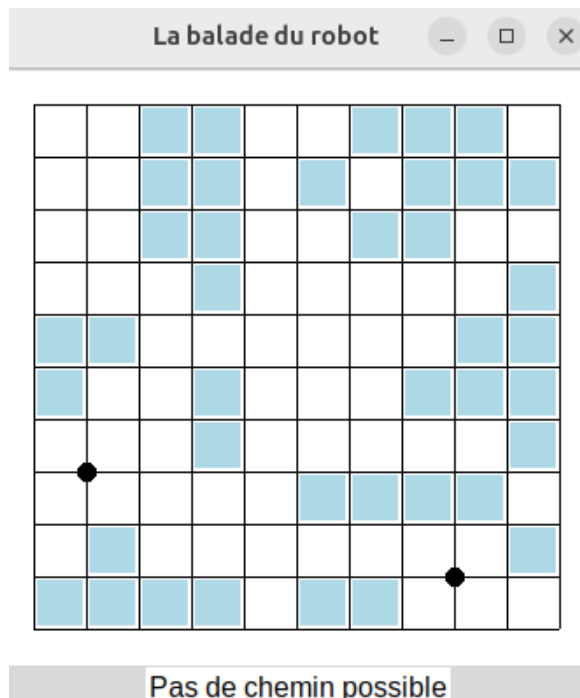
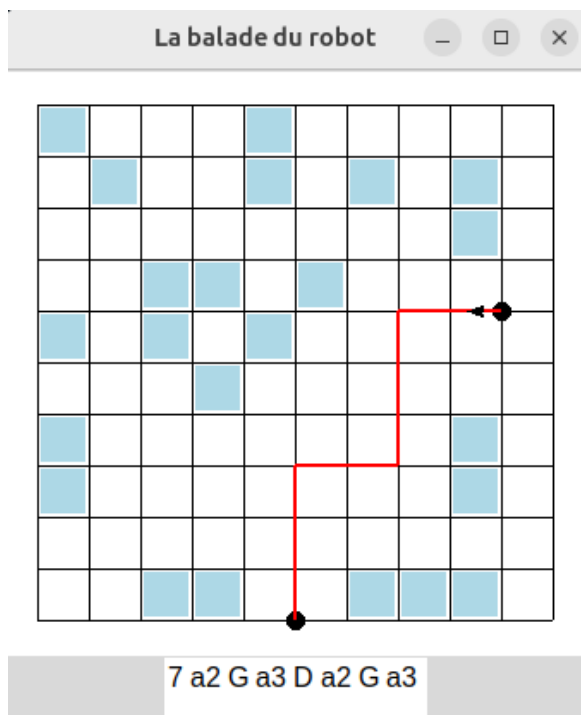


MOGPL : La balade du robot

Etudiant : **Nils Barrellon 21401602**

Github associé : <https://github.com/nbarrellon/PROJETMOGPL>



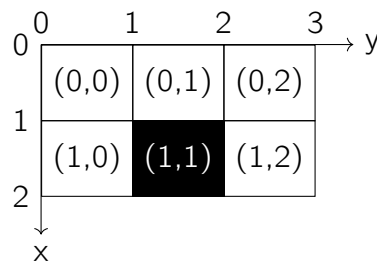
a. Formulation du problème

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 :

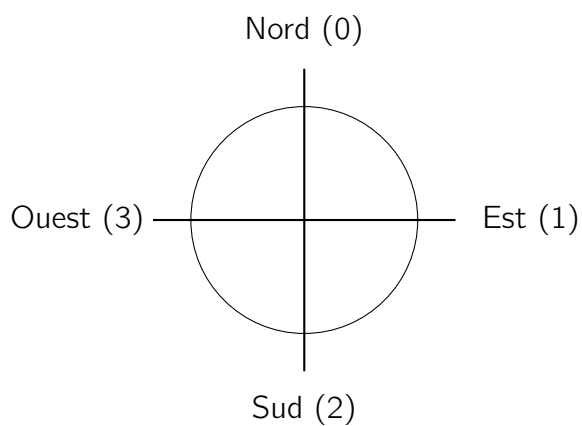
- la grille est modélisée par un graphe $G(V,E)$;
- chaque sommet V est un état du robot caractérisé par des coordonnées sur la grille qui lui sont accessibles et une orientation (N,S,E,O) ;
- Un arc traduit la possibilité pour le robot de passer d'une intersection à une autre (AVANCE) ou de tourner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

Soit la grille 2×3 suivante :

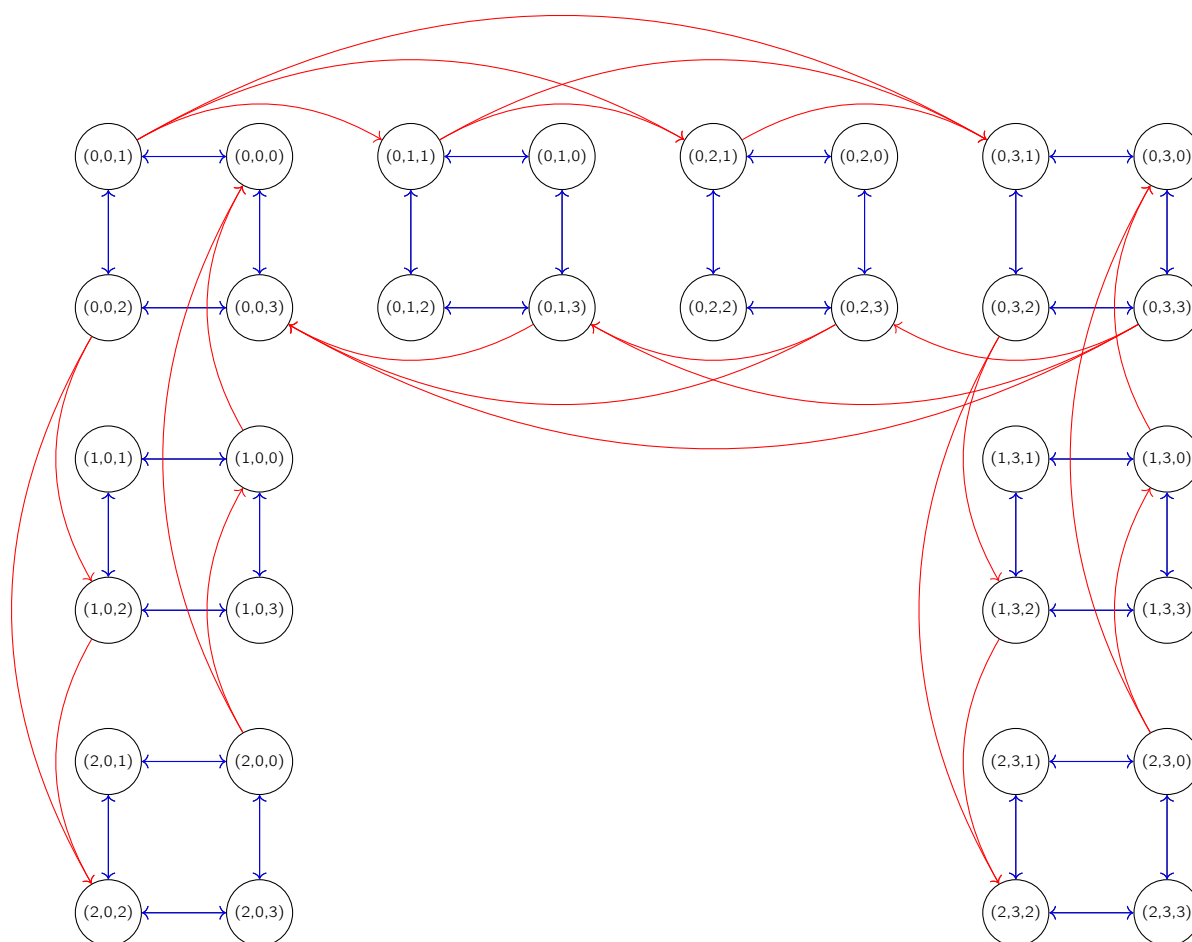


Les sommets (1,1), (1,2), (2,1) et (2,2) ne doivent pas figurer dans le graphe car le robot n'y a pas accès.

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 obtient 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 2 arêtes (pour "tourner" d'une orientation à l'autre) et 3 arêtes (pour avancer si possible vers un autre sommet) soit $|E| \leq 7(NM - 4 \times P)$.

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 présentés figures 1 et 3.

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 $N = 60$).

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 polynôme, 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.

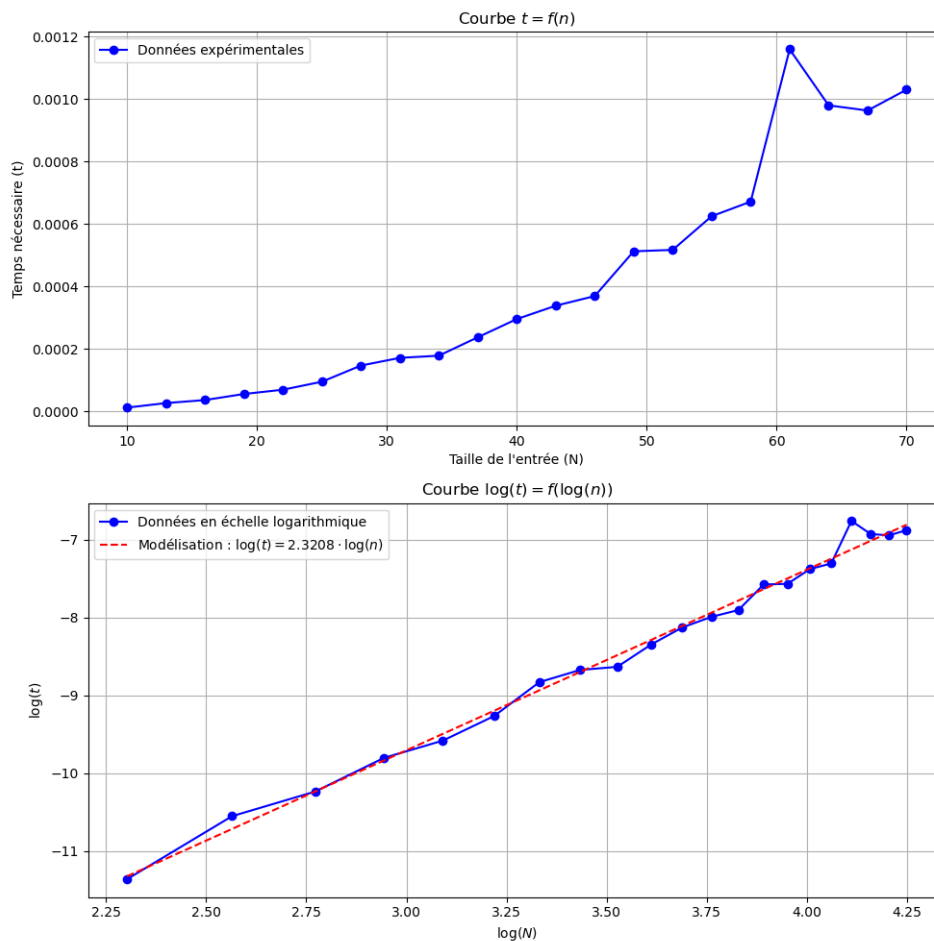


Figure 1 : Temps d'exécution en fonction de la taille de la grille pour un nombre d'obstacle constant.

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 de la taille 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 $\lim_{P \rightarrow N^2} |V| = 0$ pour une complexité qui devient alors constante.

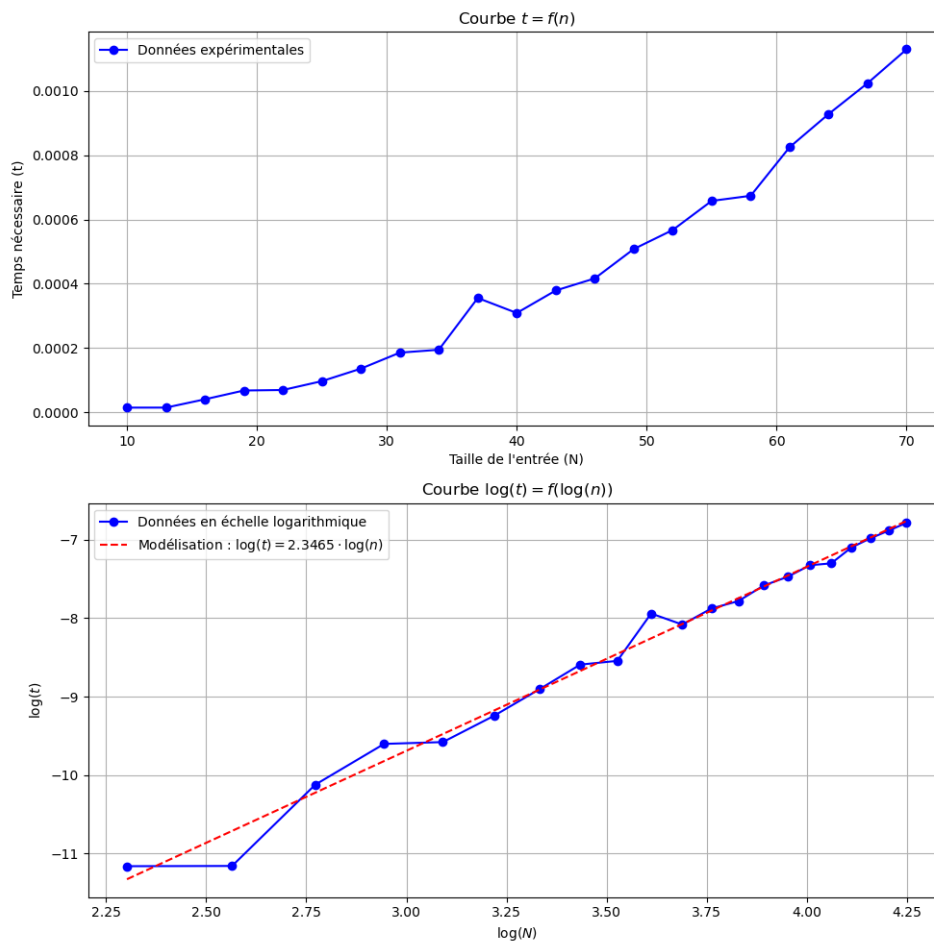


Figure 2 : Temps d'exécution en fonction de la taille de la grille pour un nombre d'obstacle constant.

La figure ?? montre comment le temps de calcul de l'algorithme temps 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.

e. Positionnement des obstacles

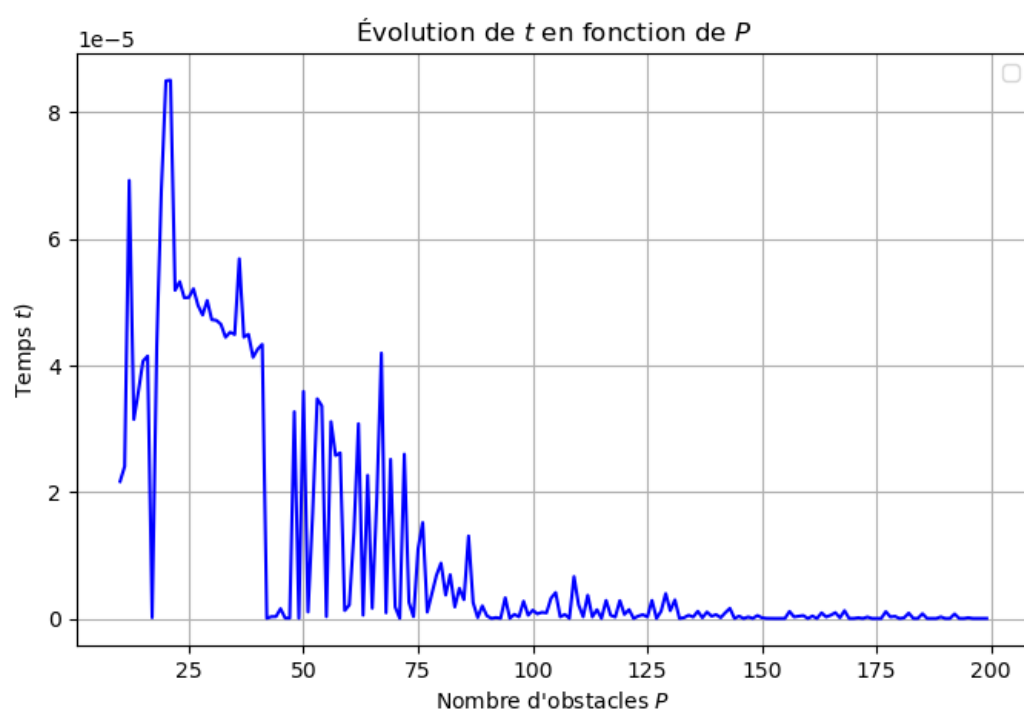


Figure 3 : Temps d'exécution en fonction du nombre d'obstacle quand la taille de la grille est constante.