Sorbonne Université

Rapport

Rendu Projet MOGPL

Élèves :
Yu Yu Chen
Emmanuel Gokana

8 décembre 2025

Table des matières

Table des figures

1 Introduction

Dans le cadre de l'UE MOGPL, nous devions développer un programme permettant de proposer un itinéraire optimal, en nombre d'actions, pour qu'un robot se déplace d'un point à un autre dans une grille composée de rails et d'obstacles, ainsi que d'effectuer des tests sur l'algorithme chargé de construire cet itinéraire.

Dans un second temps, nous devions également, à l'aide d'un programme linéaire, générer automatiquement une grille cohérente de taille (M, N) contenant un nombre d'obstacles P, ainsi que l'itinéraire optimal correspondant entre deux points choisis par l'utilisateur dans cette grille.

2 Formulation du problème de la balade du robot sous forme de graphe orienté

Pour représenter ce problème sous forme de graphe orienté nous avons décidé de concevoir un graphe qui aurait comme nœud un triplet (i, j, o) avec **i** qui représente la ligne où le robot se situe, **j** qui représente la colonne ou le robot se trouve et **o** représentant l'orientation de notre robot. Chaque nœud pointe vers les noeuds qui sont accessibles depuis ce dernier suivant la logique décrite dans le sujet du projet c'est à dire que par exemple le noeud $(0, 0, sud)$ ne pourra atteindre que les noeuds $(0, 0, est)$ et $(0, 0, ouest)$ correspondant aux rotation que le robot peut faire sur lui même. Ce noeud pourras aussi atteindre tout les noeuds à une distance inférieur ou égale à trois arcs ayant comme direction *sud*. Tout les arcs de ce noeud ont un poids de 1 correspondant au nombre de secondes nécessaire pour effectuer une action. Concernant les obstacles nous avons décidé de ne pas inclure les noeuds correspondants aux sommets entourant un obstacle dans notre graphe, ce qui fait que les arcs qui devaient normalement pointer vers ces noeuds sont naturellement absent de notre graphe. Au final nous nous retrouvons avec un graphe orienté non pondéré.

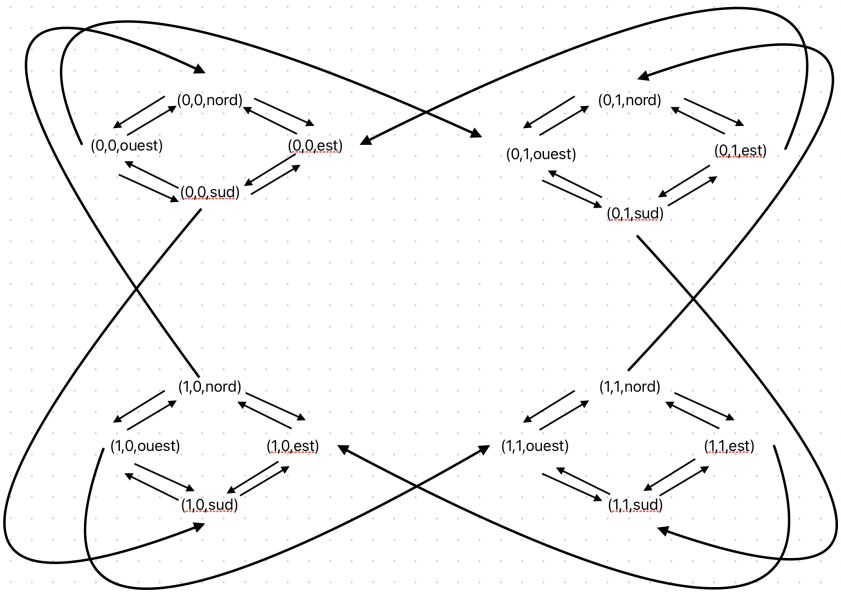

FIGURE 1 – Exemple d'un graphe représentant une grille à une case

3 Implémentation et choix de l'algorithme de plus court chemin

3.1 Implémentation du graphe

Pour implémenter notre graphe nous avons utilisé un dictionnaire qui prends pour clé un noeud du graphe sous forme de tuple (i, j, o) et comme valeur une liste de tuples $((i, j, o), c)$ ou le premier élément représente un noeud vers lequel notre noeud pointe et comme second élément la commande sous forme de chaine de caractère $c \in$ {"D","G","a1","a2","a3"} permettant d'accéder à ce noeud.

3.2 Algorithme de plus court chemin

Comme algorithme de plus court chemin nous avons choisis d'implémenter un parcours en largeur car cet algorithme nous permet de trouver le chemin comportant le moins d'arc en le sommet source et le sommet de destination et dans notre cas cela reste pertinent car notre graphe n'est pas pondéré. De plus pour un graphe $G = (V, A)$ un parcours en largeur de ce graphe se fait en $O(|V| + |A|)$.

Dans notre cas plus particulièrement pour une grille de $N \times M$ on des noeuds (i, j, o) avec $0 \leq i \leq N$, $0 \leq j \leq M$ et $o \in \{N, S, E, O\}$ donc le nombre de noeuds sera $|V| = 4 \times (N + 1) \times (M + 1)$. Concernant le nombre d'arcs sachant que nous avons 2 rotations G et D et 3 avances $(a1, a2, a3)$ on aura donc $|A| \leq 5 \times |V|$ ce qui nous fait une complexité de $\Theta(NM + NM) = \Theta(NM)$.

4 Évaluation du temps de calcul de notre algorithme de plus court chemin

4.1 En fonction de la taille de la grille

Pour évaluer les temps de calculs de notre algorithme de parcours en largeur nous avons dans un premier temps effectué nos tests sur des instances de taille différentes (M, N) qui sont $M = N = 10, 20, 30, 40, 50$ dans lesquels on a respectivement $P = 10, 20, 30, 40, 50$ obstacles et pour chaque taille d'instances nous avons généré 10 exemples et nous avons calculé le temps moyen de résolution sur ces 10 exemple voici la courbe que nous avons obtenu :

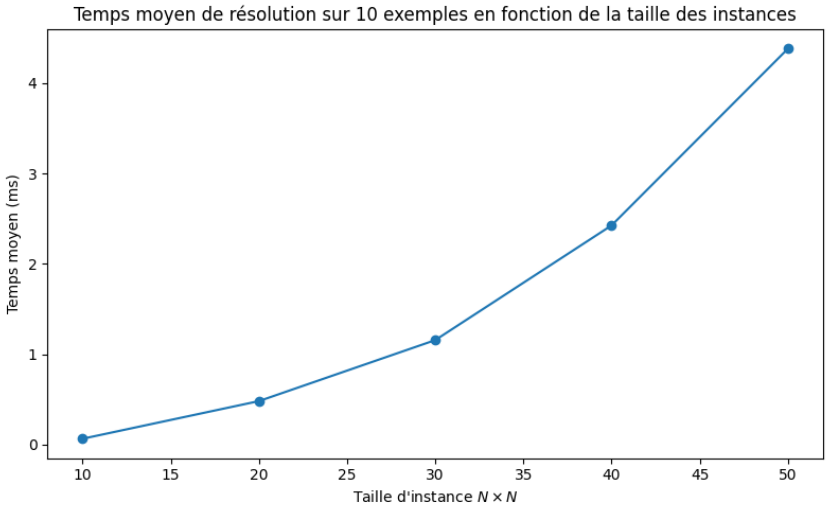

FIGURE 2 – Évolution du temps de calcul en fonction de la taille de la grille

On remarque bien sur cette courbe qu'elle adopte une forme similaire à la fonction $f(x) = x^2$ et nous avions dit précédemment que la complexité était $\Theta(MN)$ or ici $M = N$ on se retrouve donc avec une complexité de $\Theta(N^2)$ ce qui est donc cohérent avec la courbe observée.

4.2 En fonction de du nombre d'obstacles dans une grille de taille (20,20)

Pour évaluer les temps de calculs de notre algorithme en fonction du nombre d'obstacles dans une grille de taille fixe $(20, 20)$ nous avons pour chaque $P = 10, 20, 30, 40, 50$ obstacles produit 10 exemples sur lesquels nous avons calculé la moyenne du temps d'exécution voici la courbe obtenue :

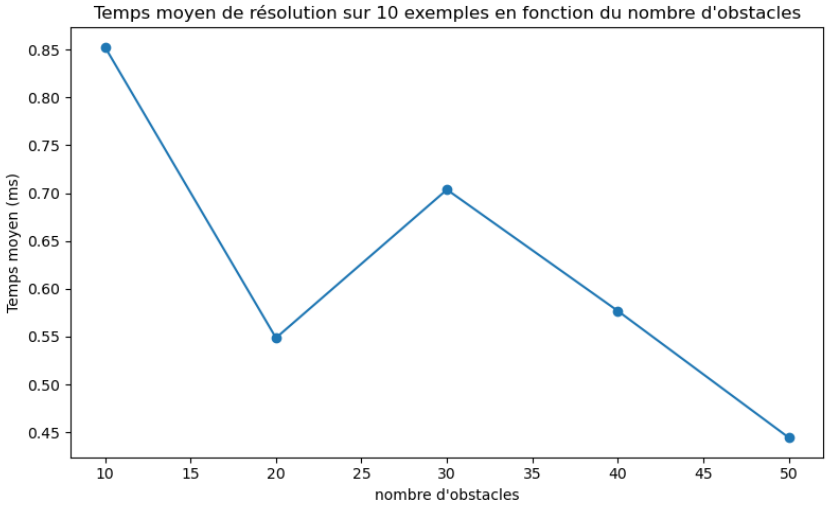

FIGURE 3 – Évolution du temps de calcul en fonction du nombre d'obstacles

Sur cette courbe on voit que plus le nombre d'obstacles augmente, plus le temps moyen de résolution lui diminue cela est due au fait que pour un grand nombre d'obstacles

le nombre de chemins possible tends à diminuer ce qui justifie une baisse des temps d'exécutions.

5 Génération aléatoire d'une grille de taille (N,M) à l'aide d'un PL

Jusqu'à maintenant pour générer nos instances nous avions utilisé un algorithme naïf qui place aléatoirement le nombre voulu P d'obstacles choisis dans la grille, mais cela faisait que ça produisait des grilles où notre robot pouvait se trouver dans des situations bloquantes, pour régler ce problème nous nous sommes aidés de la programmation linéaire pour générer une grille de manière beaucoup plus rigoureuse, nous avons donc conçût le programme linéaire suivant :

$$\text{Min} \quad \sum_{i=0}^{M}\sum_{j=0}^{N} c_{ij}x_{ij}$$

$$\text{s.c.} \begin{cases} \sum_{i=0}^{M} x_{ij} \leq \frac{2P}{N} \quad \forall j & (1) \\ \sum_{j=0}^{N} x_{ij} \leq \frac{2P}{M} \quad \forall i & (2) \\ x_{i,j-1} + x_{i,j+1} - x_{ij} \leq 1 & (3) \\ x_{i-1,j} + x_{i+1,j} - x_{ij} \leq 1 & (4) \\ \sum_{i=0}^{M}\sum_{j=0}^{N} x_{ij} = P & (5) \end{cases}$$

$$x_{ij} \in \{0,1\},$$
$$c_{ij} \in [0;1000],$$
$$i = 1,\ldots,M$$
$$j = 1,\ldots,N$$

Ce programme linéaire est composé des variables de décisions x_{ij} qui désigne le fait qu'il y ait un obstacle dans la case ij si $x_{ij} = 1$ et rien si $x_{ij} = 0$. Les coefficients c_{ij} eux désignent le nombre tiré correspondant à chaque case ce ne sont donc pas des variables de décisions.

La contrainte (1) est un ensemble de contraintes qui désigne le fait que sur une colonne le nombre d'obstacles ne doit pas excéder $\frac{2P}{N}$, la contrainte (2) elle est un ensemble de contraintes désigne le fait que sur une ligne le nombre de d'obstacles ne doit pas excéder $\frac{2P}{M}$, les contraintes (3) et (4) désigne empêchent l'apparition de la séquence 1,0,1 dans une ligne ou dans une colonne et la contrainte (5) fixe le nombre d'obstacles à P. On remarque que contrairement à d'habitude notre programme linéaire qui est un problème de minimisation à des contraintes d'infériorité mais ça ne nous pose pas de problème car la solution reste borné grâce à la contrainte (5).

6 Conclusion

Pour conclure, ce projet nous a permis de modéliser efficacement le déplacement du robot en construisant un graphe orienté adapté aux contraintes du problème, puis d'utiliser un parcours en largeur pour déterminer un itinéraire optimal en nombre d'actions. Les tests effectués ont confirmé la complexité théorique de l'algorithme et montré l'influence de la taille de la grille et du nombre d'obstacles sur les temps de calcul.

L'utilisation d'un programme linéaire pour générer automatiquement des grilles cohérentes nous a également permis d'éviter les configurations bloquantes. Ce travail nous a ainsi offert une expérience complète mêlant modélisation, algorithmique et optimisation.