Projet de mogpl - graphes et pl

Xinyi Liang, Aymeric Letaconnoux

December 2025

1 Introduction

Le projet vise à chercher un chemin optimal dans un environnement pour un robot, le robot peut se déplacer sur une grille en quatre-connexité (nord, sud, est, ouest) de une, deux ou trois intersections dans la direction où il pointe déjà, Les intersections où le robot peut se déplacer se trouve dans le coin supérieur gauche de chaque case de la grille. Nous chercherons à trouver un chemin de la position de départ du robot au point d'arrivée, qui minimise le temps pour arriver à destination. Sachant que avancer de une, deux ou trois intersections prend le même temps, de même pour tourner, toutes ces actions prennent une seconde.

2 Interprétation sous forme de graphe

Le problème proposé dans ce projet peut donc s'apparenter à un problème de plus court chemin dans un graphe orienté non-pondéré, il suffit de prendre les **états** du robot comme des sommets et les arêtes comme les **passages possibles** de l'état courant à un autre état. Ainsi nous avons :

- **Sommets** : (i, j, d) avec i l'indice de la ligne, j l'indice de la colonne et d l'orientation nord sud est ouest du robot.
- **Arêtes** : Transition valide entre deux états.
- **Voisins** : (i±k, j, d), (i, j±k, d), (i, j, d') avec k ∈ {1, 2, 3}, d l'orientation de départ et d' la nouvelle orientation différente de d, d et d' ∈ {N, E, S, W} et $0 \leq$ i±k $\leq$ N-1 et $0 \leq$ j±k $\leq$ M-1

En théorie il y a donc cinq voisins maximum par états, soit le robot tourne à gauche ou à droite, ou alors il avance dans sa direction de 1, 2 ou 3 intersections. Si on prend l'exemple de la consigne, à l'état (7, 2, S) le robot a trois voisins : il peut soit tourner vers l'ouest, soit tourner vers l'est, ou alors avancer d'une case dans sa direction, il ne peut pas avancer de 2 ou 3 cases, sinon il se retrouverait en dehors de la grille.

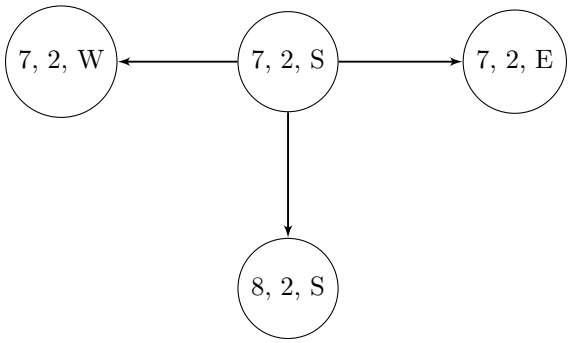

Figure 1: État de départ et ses 3 voisins

3 Algorithme et complexité

Maintenant que nous avons établis le graphe sur lequel nous allons travailler il ne nous reste plus qu'à résoudre le problème posé, il s'agit d'une recherche de plus court chemin dans un graphe non pondéré, pour résoudre ce problème de manière efficace l'algorithme du **BFS** (Breath-First Search ou Parcours en largeur) est le plus approprié. L'algorithme BFS parcourt le graphe en visitant tous les voisins du sommet courant avant de passer aux voisins des sommets du niveau supérieur. De cette manière on aura visité tous les sommets d'un niveau inférieur avant de passer aux niveaux supérieur, nous permettant ainsi de trouver le plus court chemin entre deux sommets. Nous avons ajouté une condition d'arrêt au BFS pour qu'il n'explore pas le reste du graphe si nous avons déjà trouvé le sommet de fin.

3.1 Complexité

La complexité du BFS est O(|V|+|E|) avec |V| le nombre de sommets et |E| le nombre d'arêtes. Cependant, le BFS utilise deux autres fonctions pour récupérer les voisins du sommet courant : *clear_path()* et *get_neighbors()*.

- ***clear_path()*** a une complexité constante O(1), en effet il y a dans la fonction deux boucles l'une à la suite de l'autre qui vérifient toutes deux si les cases entre le sommet courant et le voisin potentiel ne sont pas des obstacles, la distance maximale entre ces deux sommets est connue quelque soit la taille de la grille et ne dépasse pas 3 (déplacement maximal du robot à chaque mouvement).
- ***get_neightbors()*** a la même complexité, la boucle ne dépasse jamais 3 itérations et elle utilise *clear_path()* en O(1), donc *get_neighbors()* est aussi en O(1).

Par conséquent, *get_shortest_path()* qui est une implémentation du BFS utilisant *get_neighbors()* a une complexité en **O(|V|+|E|)**.

4 Essais

4.1 Grilles variables, obstacles fixes

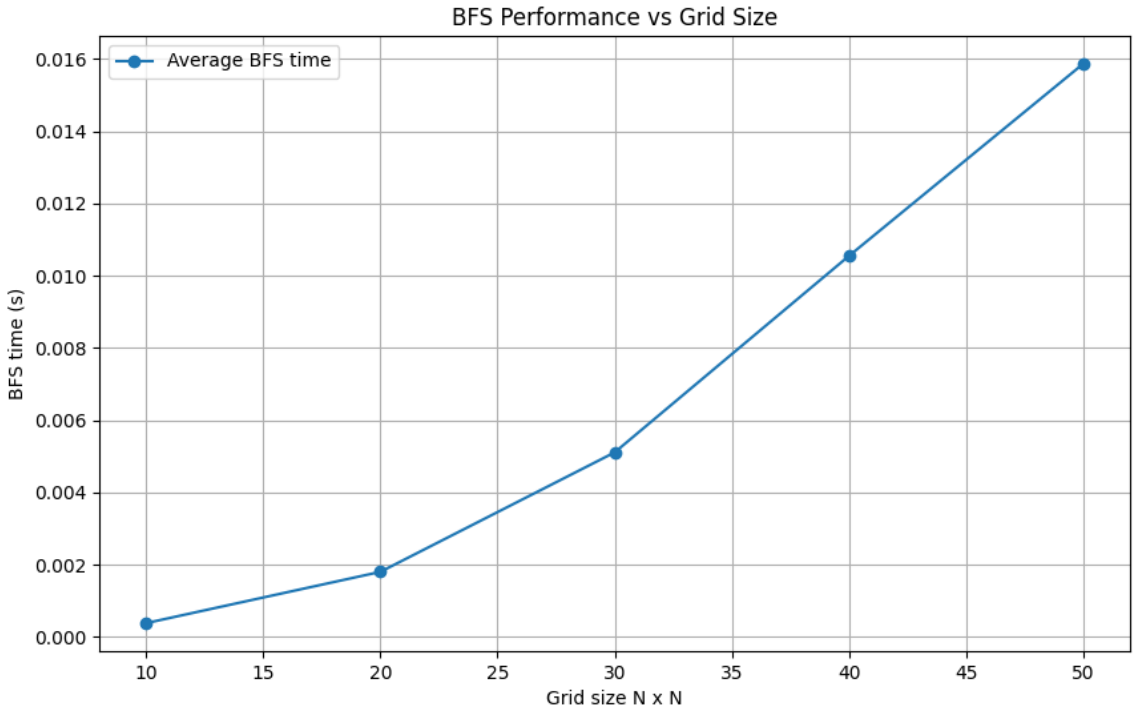

Figure 2: Temps de résolution en fonction de la taille de la grille

On voit que le temps de résolution suit une évolution croissante avec l'augmentation de la taille de la grille. Ce qui est cohérent avec la complexité du BFS, qui dépend de V et E, en effet si la taille de la grille augmente alors le nombre de sommets et d'arêtes augmentera aussi.

4.2 Grille fixée, obstacles variables

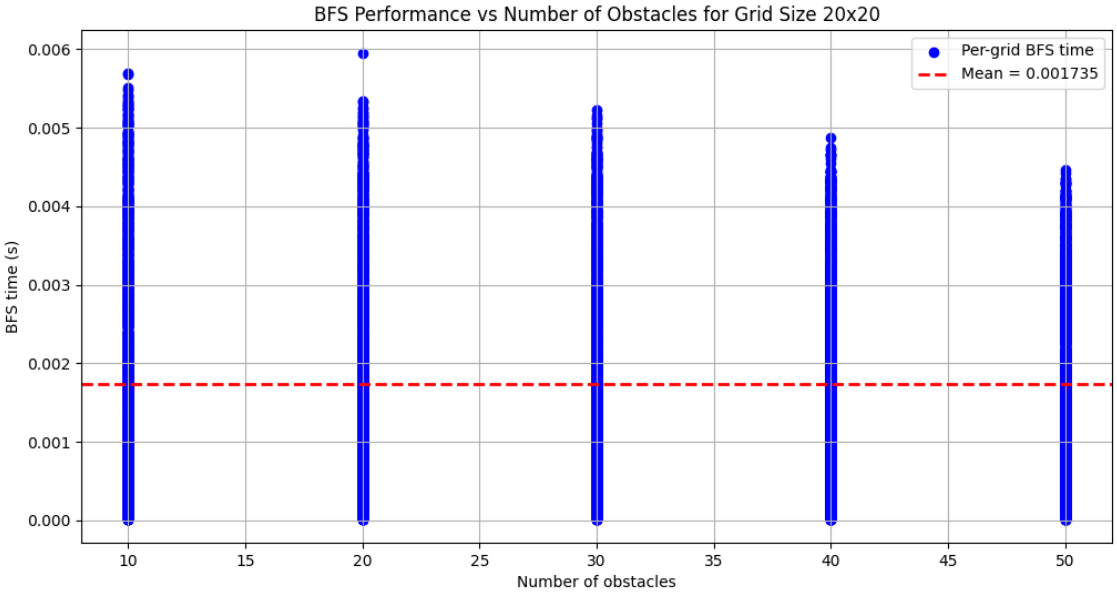

Figure 3: Temps de résolution en fonction du nombre d'obstacles

Ici le temps de résolution ne semble pas augmenter, au contraire il semble diminuer légèrement. Là aussi l'explication vient de la complexité du BFS, en effet si l'on ne change pas la taille de la grille alors |V| n'augmentera pas, par contre si l'on ne considère que les arêtes possible comme dans notre cas alors |E| diminuera si on augmente le nombre d'obstacle sans changer la taille de la grille.

5 Programmation linéaire

Nous modélisons le problème d'optimisation consistant à engendrer aléatoirement une grille de taille (M, N) contenant exactement P obstacles, tout en respectant les contraintes imposées. La modélisation retenue est la suivante.

Variables de décision.

$$x_{ij} = \begin{cases} 1 & \text{si la case } (i,j) \text{ contient un obstacle,} \\ 0 & \text{sinon.} \end{cases}$$

Domaine.

$$x_{ij} \in \{0,1\} \qquad \forall i \in \{1,\dots,M\},\ \forall j \in \{1,\dots,N\}.$$

Fonction objectif. Chaque case reçoit un poids aléatoire $w_{ij} \in \{0,\dots,1000\}$. L'objectif est de minimiser la somme des poids des cases choisies :

$$\min \sum_{i=1}^{M} \sum_{j=1}^{N} w_{ij}\, x_{ij}.$$

Contraintes.

1. **Nombre total d'obstacles.**
$$\sum_{i=1}^{M}\sum_{j=1}^{N} x_{ij} = P.$$

2. **Maximum $2P/M$ obstacles par ligne.**
$$\sum_{j=1}^{N} x_{ij} \leq \frac{2P}{M} \qquad \forall i.$$

3. **Maximum $2P/N$ obstacles par colonne.**
$$\sum_{i=1}^{M} x_{ij} \leq \frac{2P}{N} \qquad \forall j.$$

4. **Interdiction de la séquence 101 sur une colonne.**
$$x_{i-1,j} + x_{i+1,j} - x_{ij} \leq 1 \qquad \forall i = 2, \ldots, M-1,\ \forall j.$$

5. **Interdiction de la séquence 101 sur une ligne.**
$$x_{i,j-1} + x_{i,j+1} - x_{ij} \leq 1 \qquad \forall i,\ \forall j = 2, \ldots, N-1.$$

6 conclusion

Dans ce projet, nous avons modélisé le déplacement du robot comme un problème de plus court chemin dans un graphe orienté non pondéré. L'algorithme BFS s'est révélé adapté pour trouver un chemin optimal grâce à sa complexité linéaire et à son parcours efficace des états atteignables. Les résultats expérimentaux confirment que le temps de calcul augmente avec la taille de la grille mais diminue lorsque le nombre d'obstacles réduit les transitions possibles. Nous avons également formulé un modèle de programmation linéaire permettant de générer automatiquement des grilles en utilisant les entrées fournies par l'utilisateur via le terminal, tout en respectant les contraintes imposées.