LA BALADE DU ROBOT

PROJET

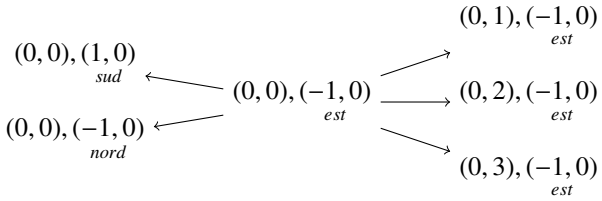
Schwarz Lucie
Rhouzlane Elias

Université Pierre et Marie Curie
M1 Androide - MOGPL

Décembre 2015

Question A

On cherche à minimiser le temps de transport d'un robot dans une entreprise contenant certains obstacles, ce robot n'a qu'un nombre limité de déplacement. On peut donc modéliser ce problème comme une recherche de plus court chemin dans un graphe orienté, où les noeuds sont les localisations possibles du robot (localisé par deux coordonnées ainsi que l'orientation), et les arcs les déplacements possibles reliant un noeud de départ A à un noeud d'arrivée B.



Exemple de noeud et ses fils représentant une position et une orientation dans l'espace disponible du robot.

Les déplacements entre noeuds peuvent représenter un changement d'orientation à gauche ou à droite ou un déplacement d'une, deux ou trois cases en avant si et seulement si cette case existe et n'est pas bloqué par un obstacle. Le poids des arcs étant fixé à 1 et chaque déplacement étant ainsi du même coût, il est idéal d'utiliser un parcours en largeur pour trouver notre meilleur chemin.

Question B

Notre algorithme est un algorithme de type parcours en largeur, à partir d'un noeud on évalue tous ses noeuds atteignables en un seul arc, puis on parcourt tous les noeuds atteignables en un seul arc de ses fils et ainsi de suite. Les obstacles n'étant pas explorés, la complexité est donc en $O(N + M - \mathrm{O})$ avec :

- N : le nombre de lignes
- M : le nombre de colonnes
- O : le nombre d'obstacles

Durant le parcours du graphe, dès qu'un noeud correspondant à la position d'arrivée est trouvé (quelque soit l'orientation de ce noeud), le chemin qui a été pris est à coup sûr le plus court (le parcours se faisant en largeur). Il y a donc quatres noeuds d'arrivée possibles (une position et quatres orientations).

L'implémentation de l'algorithme s'est fait en Python 3, ce choix est notamment motivé par son orientation objet et la grande flexibilité du langage tout en restant facile à lire et à produire.

La génération des instances s'effectue avec la commande suivante:

```
python3 instances.py N=10 M=10 O=10 S=5 nom_du_fichier
```

avec N, M, O et S le nombre de ligne, de colonne, d'obstacle et d'instance, puis le nom du fichier (sans extension). Le fichier d'entrée est ensuite stocké dans le dossier ../data/inputs/ sous le nom 'nom_du_fichier'.dat.

La résolution des instances stockés dans un fichier d'entrée (dans le dossier ../data/inputs/) se fait avec la commande suivante:

```
python3 main.py nom_du_fichier
```

Le fichier résultat est ensuite stocké dans le dossier ../data/outputs/ sous le nom 'nom_du_fichier_output'.dat.

Question C

Ici nous avons généré des instances avec $N = M = 10, 20, 30, 40, 50$ avec chaque fois un nombre d'obstacles fixé ($N = M =$ O). Les instances ont été stockés dans un fichier d'entrée et les résultats dans un fichier de résultats. Pour chaque valeur de N on a ensuite tirer 10 instances aléatoirement qu'on a reporté sur la courbe suivante les temps moyen d'exécution. (fig.1, fig.2)

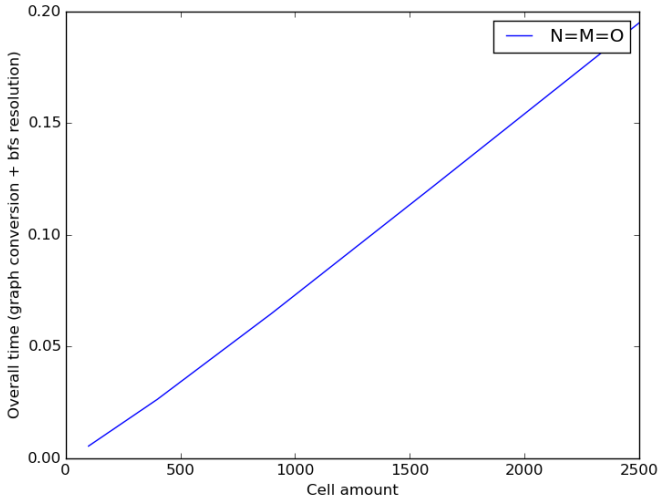


Figure 1: Évaluation du temps de calcul de l'algorithme en fonction de la **taille de la grille (en secondes)**

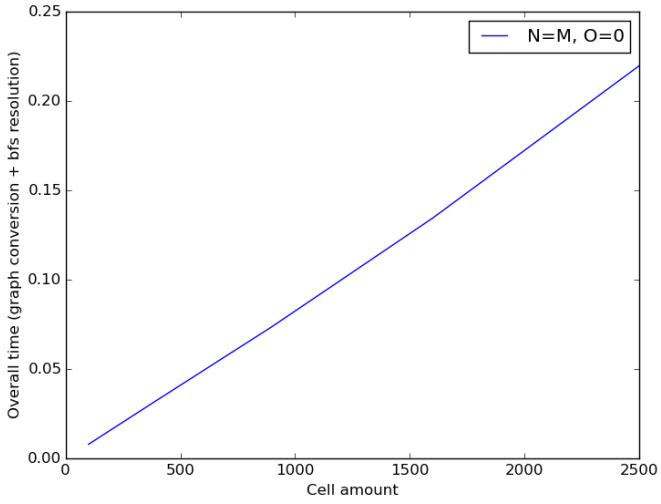


Figure 2: Évaluation du temps de calcul de l'algorithme en fonction de la **taille de la grille (en secondes)**

Les deux graphiques sont les temps moyen d'éxecution en fixant le nombre d'obstacle à 0 et l'autre en fixant le nombre d'obstacle aux mêmes valeurs N et M.

On observe que l'algorithme est linéaire au nombre de case dans la grille. En effet, la génération du graphe se fait en $o(4 * N)$ avec N le nombre de case de notre grille et le parcours en largeur se limite aux noeuds déjà visité. De plus, grâce au deuxième graphique nous avons un petit indice qui nous laisse suggérer que le temps moyen d'éxecution de l'algorithme est inversement proportionnel au nombre d'obstacles pour une même taille de grille. Nous y reviendrons en détail dans la question D.

Question D

Nous avons ensuite effectué des essais numériques pour évaluer le temps de calcul de notre algorithme en fonction du nombre d'obstacles présents. Pour une grille de taille 20×20 nous avons généré des instances avec 10, 20, 30, 40 puis 50 obstacles. Les instances ont été stockés dans un fichier d'entrée et les résultats dans un fichier résultats. Pour chaque valeur de nombre d'obstacles nous avons tiré 10 instances aléatoirement dont on a reporté sur la courbe suivante les temps moyen d'exécution. (fig.3)

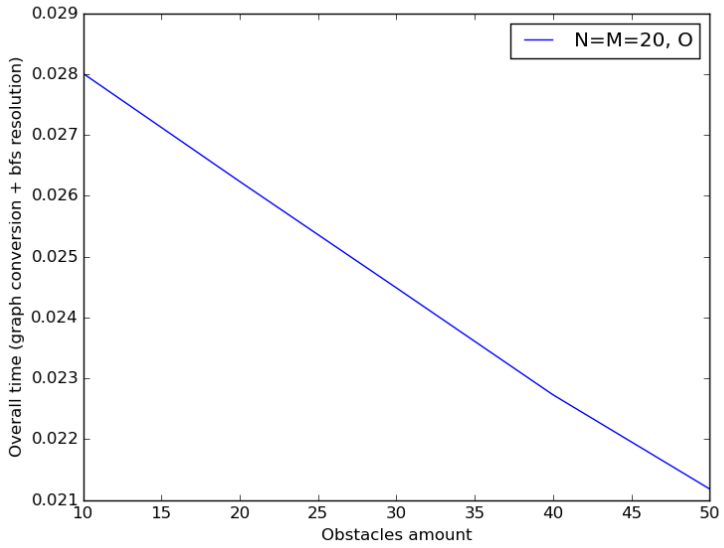


Figure 3: Évaluation du temps de calcul de l'algorithme en fonction du **nombre d'obstacles présents (en secondes)**

Nous remarquons ici que pour une taille d'instance fixe, le nombre d'obstacles présents réduit le temps moyen d'éxecution. Cela peut s'expliquer par le fait que le nombre d'obstacle représente aussi le nombre de noeud qui n'ont aucune relation avec d'autres noeuds, ils sont donc orphelins. Ainsi, aucun chemin ne pourra passer par ces noeuds, ils ne sont donc pas à explorer. Le temps d'execution etait lié au parcours des noeuds, il sera donc réduit proportionnellement. De plus lors de la création du graphe, on ne cherchera pas les arcs potentiels de ce noeud, il sera directement ignoré. Le temps de calcul est donc inversement proportionnel et linéaire au nombre d'obstacle.

Question E

Nous proposons une interface de type web permettant à l'utilisateur de choisir la taille de la grille, le nombre d'obstacles et l'affichage de la solution obtenue. L'interface utilise le framework opensource Django (fig.4). Nous avons ainsi configuré une *webapp* dynamique travaillant de concert avec nos algorithmes de génération et de résolution du problème. (fig.5, fig.6)

Figure 4: Django est un framework web python de haut-niveau

Après avoir configuré un environnement python avec Django, il est possible de lancer un serveur émulant notre programme via la commande suivante:

```
python3 manage.py runserver
```

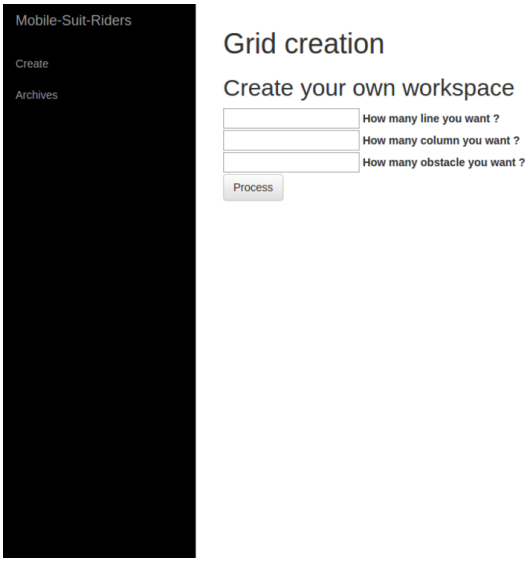


Figure 5: Interface de génération de grille, entrées pour le nombre de lignes, colonnes et obstacles

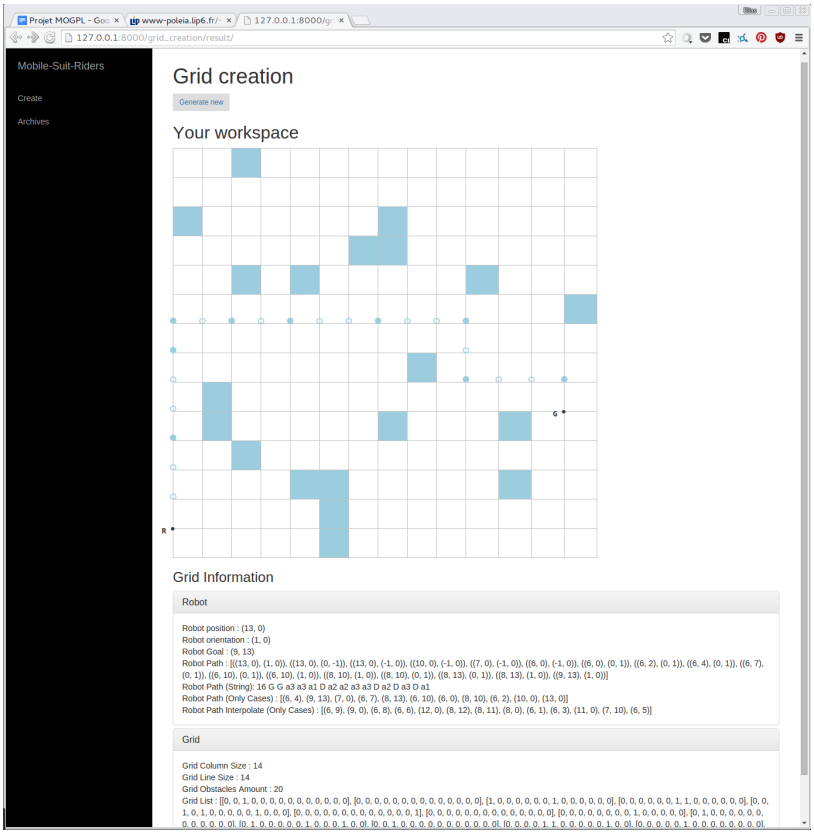


Figure 6: Grille automatiquement générée par le programme, l'algorithme trouve le plus court chemin et l'affiche