TP1 – ALGORITHMES DE RECHERCHE

I - OBJECTIF

L'objectif de ce premier TP est d'implémenter différents algorithmes de recherche et d'explorer leur performance sur des exemples simples.

II - EVALUATION

Le travail préparatoire est récupéré en début de séance par l'encadrant. Les codes implémentés lors du TP et un compte-rendu électronique, contenant les réponses aux questions posées de l'énoncé ainsi que les commentaires sur les algorithmes et leurs performances, sont récupérés en fin de séance. Afin de faciliter cette étape, regroupez tous les documents nécessaires dans un dossier à votre nom.
Le travail réalisé sera évalué sur la qualité de la préparation, la progression lors de la séance, la clarté des codes (pensez à les commenter) et la pertinence du compte-rendu.

III - Préparation Théorique

1) COMPLEXITÉ EN ESPACE

Les figures 1 et 2 présentent deux arbres de recherche avec un nombre de noeuds identique mais des structures différentes.

Q1. Quel algorithme de recherche non-informé est le plus adéquat pour chaque graphe selon le critère de complexité en espace ? Justifiez votre réponse.

2) REPRÉSENTATIONS DE DONNÉES

Afin de pouvoir appliquer des algorithmes de recherche, les informations concernant le graphe étudié doivent être stockées de manière adéquate. A cette fin, une matrice d'incidence peut, par exemple, être utilisée.

Q2. Déterminez la matrice d'incidence associée au graphe de la figure 3.

Q3. Quelles autres structures de données pourraient également être utilisées pour stocker ces informations ? Comparez-les avec la matrice d'incidence.

IV - CAS PRATIQUE

Le fichier ***TP1.m*** génère une interface graphique correspondant à un espace 2D discret dans lequel un but, des obstacles et la position initiale d'un robot peuvent être définies. Un cout ***k=1*** est attribué à chaque déplacement horizontal ou vertical. Un cout $\mathbf{k=\sqrt{2}}$ est attribué aux déplacements en diagonale.
L'exécution de **TP1.m** génère les variables suivantes :

- ***Plan*** (Matrice) : représentation de l'espace 2D selon la convention suivante :
  - Espace libre : 1
  - Obstacle : -1
  - But : 2
  - Position initiale : 0

- ***NodeList*** (Structure) : structure d'un graphe où chaque nœud représente une case de l'espace 2D. Les voisin de chaque noeud ***n*** sont stockés dans le champ ***NodeList (n).Neigh***. Les coûts entre le noeud **n** et ses voisins sont stockés dans le champ ***NodeList (n).K***.
  (Remarque : pour la suite du TP, il est possible de définir de nouveaux champs dans la structure.)

- ***Nodes*** *(Matrice)* : matrice d'incidence du graphe contenant les mêmes informations que ***NodesList*** sous forme matricielle. L'élément ***Nodes(i,j)*** représente le cout ***k*** pour aller du nœud ***i*** au nœud ***j***. Le coût est nul si les noeuds **i** et **j** ne sont pas reliés.

- ***RootNode*** *et* ***TargetNode*** (scalaires) : nœuds correspondant respectivement à la position initiale et au but.

- ***Map_plan2node*** (Matrice) : matrice contenant le numéro du noeud associé à chaque case de l'espace 2D.

1) ALGORITHMES NON-INFORMÉS

Q4. Écrire la fonction ***RechercheEnLargeur.m*** effectuant une recherche en largeur (algorithme « Breadth First search ») d'un chemin permettant d'atteindre le but à partir de la position initiale. Cette fonction aura comme entrées le graphe étudié (***NodeList*** *ou* ***Nodes***), le nœud associé à la position initiale (**RootNode**) et le nœud associé au but (**TargetNode**). Elle renverra le nombre de noeuds parcourus et le chemin trouvé dans le cas où le but a été atteint. Elle renverra un message indiquant l'échec dans le cas contraire.

Q5. En respectant la structure de fonction précédente, écrire une seconde fonction RechercheEnProfondeur.m effectuant une recherche en profondeur (algorithme « Depth First Search »).

2) ALGORITHMES INFORMES

Nous supposons que la distance entre une case quelconque et la case but peut être estimée par une ligne droite dans l'espace 2D.

Q6. Écrire la fonction **Astar.m** effectuant une recherche selon l'algorithme A*. Cette fonction aura comme entrées le graphe étudié (***NodeList*** *ou* ***Nodes***), le nœud associé à la position initiale (**RootNode**), le nœud associé au but (**TargetNode**) et la matrice ***Map_plan2node*** (utile pour calculer la distance entre deux cases). Elle renverra une liste des noeuds explorés ainsi que le chemin trouvé dans le cas où le but est atteint. Elle renverra un message indiquant l'échec dans le cas contraire.

La fonction **AnimatePath.m** permet de visualiser séquentiellement le parcours découvert entre la case initiale et le but. Appliquez-la aux chemins déterminés avec les algorithmes précédemment implémentés.

Influence de l'heuristique :

L'influence de l'heuristique peut être explorée en remplaçant le calcul de distance euclidienne dans l'espace 2D par la « distance de Manhattan ».

Q7. Apportez les modifications nécessaires à ***Astar.m*** pour effectuer la recherche en prenant en compte la distance de Manhattan entre les cases. Nommez cette nouvelle fonction **AstarMan.m** et comparez les deux heuristiques.

Effet de coûts variables :

Si le robot se déplaçait sur un terrain accidenté, les variations de relief et d'adhérence pourraient être représentées par des coûts de déplacements différenciés.
Exécutez le script **CoutVariable.m** pour créer de nouveaux coûts non-uniformes dans les variables ***Nodes*** et ***NodeList(n).K.***

Q8. Appliquez la fonction **Astar.m** avec des coûts de déplacement non-uniformes. Analysez l'influence de ce changement en comparant les résultats avec ceux obtenus précédemment.

Complexité en temps :

Les commandes ***tic*** et ***toc*** permettent d'estimer le temps d'exécution de lignes d'instruction Matlab :

>> tic, [sorties] = MaFonction(entrees); toc;

La ligne précédente affiche par exemple : « *Elapsed time is 0.000006 seconds ».*

Le temps d'exécution peut également être stocké dans une variable comme suit:

>> tic ; [sorties] = MaFonction(entrees) ; temps_execution_1=toc ;

Q9. Déterminez empiriquement l'algorithme le plus rapide pour trouver un chemin jusqu'au but. (Pensez à varier les scénarios afin de généraliser les résultats).

Complexité en espace :

Les matrices d'incidence associées aux arbres des figures 1 et 2 sont stockées dans les variables **tree1.mat** et **tree2.mat**. Chargez ces deux matrices et utilisez (si nécessaire) la fonction **Matrix2List.m** pour obtenir les listes correspondantes.

Q10. Utilisez les fonctions ***RechercheEnLargeur.m*** *et* ***RechercheEnProfondeur.m*** pour estimer empiriquement l'algorithme de moindre complexité en espace pour chacun des deux graphes.

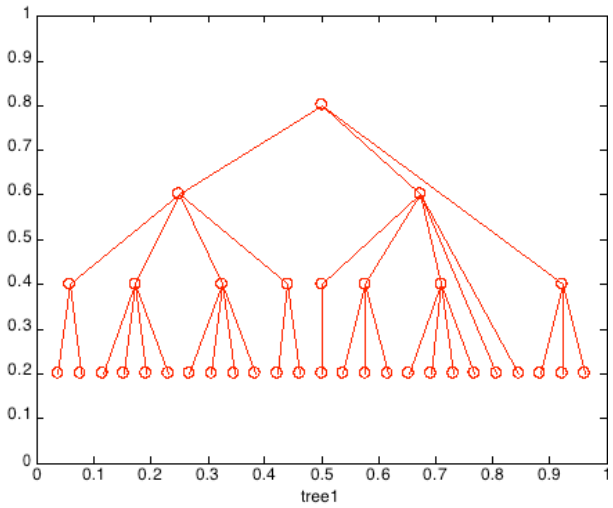

Figure 1 : tree1

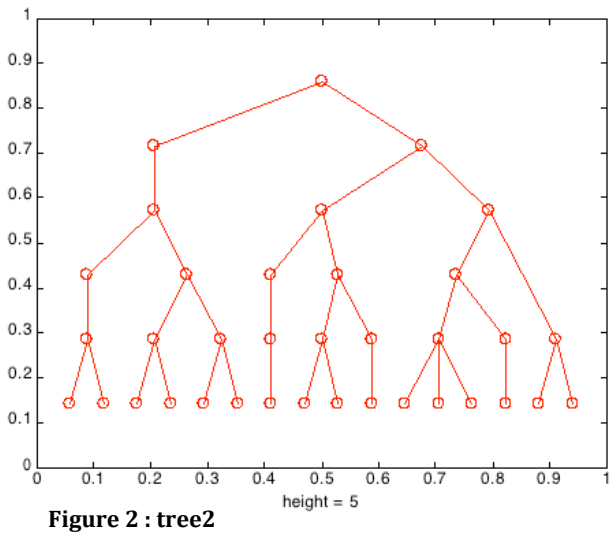

Figure 2 : tree2

Figure 3 :