Analyse et Comparaison des Algorithmes de Recherche de Chemin : Dijkstra et A* PARTIE B

Ben Slama Sana

27/12/2024

Introduction

Dans ce rapport, nous analysons les algorithmes de Dijkstra et A* pour la recherche du plus court chemin dans un graphe avec connexité étendue (8 voisins). Nous discutons des structures de données utilisées pour optimiser leur complexité, justifions le choix des heuristiques, et évaluons leurs performances respectives. Enfin, nous proposons une réflexion sur les cas d'utilisation réels de ces algorithmes.

Structures de Données Utilisées

1. Graphe

Le graphe est représenté à l'aide de :

— Une liste de sommets (`vertexlist`), où chaque sommet contient des informations comme son coût cumulatif (`timeFromSource`), son prédécesseur (`prev`), et sa valeur heuristique (`heuristic`).

— Chaque sommet contient une liste d'adjacence stockant les arêtes vers ses voisins, avec les poids calculés selon les règles définies dans l'énoncé.

2. File de Priorité (PriorityQueue)

— Utilisée pour extraire efficacement le sommet avec le coût le plus faible (ou la plus petite valeur de `f = g + h` pour A*).

— Complexité : pour l'insertion et l'extraction.

Justification du Choix de l'Heuristique

Heuristique de Manhattan

L'heuristique de Manhattan est adaptée aux graphes où les déplacements s'effectuent uniquement de manière horizontale ou verticale. Elle calcule la distance entre deux points en considérant uniquement la somme des différences absolues entre leurs coordonnées respectives.

Cependant, dans ce cas particulier, les déplacements diagonaux sont autorisés. Par conséquent, cette heuristique devient inappropriée car elle ne reflète pas fidèlement les coûts réels des déplacements possibles dans le graphe.

Heuristique Euclidienne

L'heuristique Euclidienne est mieux adaptée aux graphes permettant des déplacements diagonaux. Elle calcule la distance en ligne droite entre deux points à l'aide de la formule de la distance Euclidienne.
Cette heuristique est admissible (elle ne surestime jamais le coût réel) et consistante, ce qui garantit les propriétés de convergence et l'efficacité de l'algorithme A*.

Illustration de la Connexité

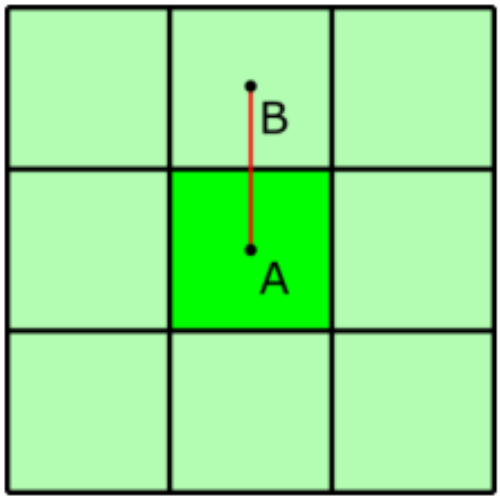

FIGURE 1 – Illustration de la grille avec 8 voisins.

Dans la grille ci-dessus, les déplacements diagonaux entre les cases vertes

claires sont possibles. Cela justifie l'utilisation de l'heuristique Euclidienne pour représenter les coûts avec précision.

Performance des Algorithmes

Algorithme de Dijkstra

— Explore tous les sommets atteignables depuis la source avant de trouver le chemin optimal.
— Complexité temporelle : , où est le nombre de sommets et le nombre d'arêtes.
— Utilise uniquement `timeFromSource` sans heuristique.

Algorithme A*

— Réduit l'exploration inutile grâce à l'utilisation d'une heuristique admissible.
— Complexité temporelle similaire à celle de Dijkstra, mais réduit le nombre de nœuds explorés.
— Performances observées dans notre implémentation : **A* explore environ 5 nœuds de moins que Dijkstra pour le même chemin.**

Cas Réels d'Utilisation

Algorithme de Dijkstra

— Adapté aux graphes sans heuristique spécifique ou aux scénarios où tous les chemins doivent être explorés (par exemple, les réseaux routiers avec coûts fixes).
— Idéal pour calculer des distances entre tous les paires de sommets (algorithme de Floyd-Warshall).

Algorithme A*

— Utilisé dans les systèmes de navigation (GPS), les jeux vidéo, et la robotique où une heuristique peut être exploitée pour guider la recherche.
— Convient particulièrement aux scénarios où la rapidité est cruciale et où une estimation des coûts est disponible.

Pourquoi l'heuristique de Manhattan est plus efficace ici

Dans ce cas particulier, bien que l'heuristique de Manhattan soit moins précise que l'heuristique Euclidienne pour représenter les coûts réels dans un graphe autorisant des déplacements diagonaux, elle est plus efficace en termes de performances (nombre de nœuds explorés). Voici pourquoi :

L'heuristique de Manhattan repose sur des opérations arithmétiques simples :

$$h(x_1, y_1, x_2, y_2) = |x_1 - x_2| + |y_1 - y_2|$$

Elle calcule la somme des distances absolues entre les coordonnées des deux points (x_1, y_1) et (x_2, y_2). Cette heuristique est adaptée aux déplacements exclusivement horizontaux et verticaux, mais elle peut sous-estimer les coûts réels dans les cas où des déplacements diagonaux sont autorisés. Cela peut conduire à une exploration plus large mais moins coûteuse en termes de calculs mathématiques par nœud. Cette sous-estimation peut inciter l'algorithme A* à privilégier des chemins plus directs sans explorer trop de voisins immédiats. L'heuristique de Manhattan explore moins les nœuds inutiles pour atteindre une solution, car sa nature simpliste peut parfois être suffisante pour converger rapidement, même si elle n'est pas parfaitement adaptée aux déplacements diagonaux. L'heuristique de Manhattan tend à être plus optimiste (sous-estime la distance réelle) par rapport à l'heuristique euclidienne. Cela signifie qu'elle peut souvent trouver des chemins plus courts plus rapidement, ce qui réduit le nombre de nœuds explorés.

En revanche, l'heuristique Euclidienne implique des calculs plus complexes (racine carrée) :

$$d_{\text{Euclidienne}} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Elle offre une estimation plus précise des coûts réels dans les graphes où les déplacements diagonaux sont autorisés et garantit que l'algorithme A* reste optimal et explore les chemins les plus prometteurs. Elle est idéale pour les graphes où la distance en ligne droite est représentative du coût réel (comme les déplacements dans un espace continu ou une carte géographique avec diagonales).

L'heuristique Euclidienne reste cependant préférable lorsqu'une estimation précise et cohérente des coûts est cruciale pour garantir un comportement optimal et réaliste de l'algorithme A*. Bien qu'elle soit plus précise pour les mouvements diagonaux, elle peut explorer plus de nœuds en raison de sa nature plus conservatrice. Dans notre cas, nous avons utilisé l'heuristique Euclidienne pour prendre en compte les mouvements diagonaux bien que l'heuristique est moins efficace que la Manhattan.

Exécutions sur différentes cartes

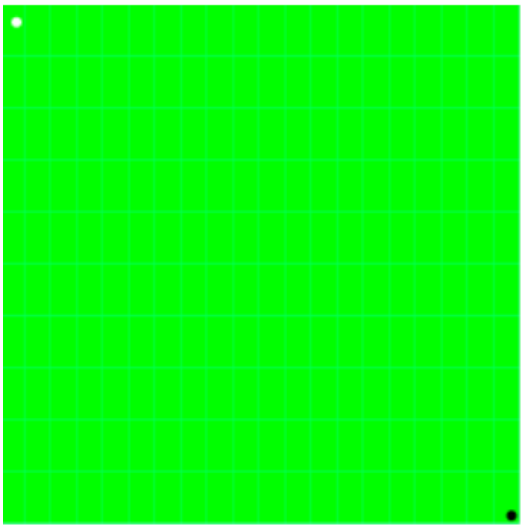

FIGURE 2 – Graphe 1

```
Done! Using Dijkstra:
    Number of nodes explored: 379
    Total time of the path: 26.870057685088817
```

FIGURE 3 – Résultats d'exécution avec Dijkstra

```
Done! Using A* (Euclide):
    Number of nodes explored: 19
    Total time of the path: 26.870057685088817
```

FIGURE 4 – Résultats d'exécution avec A*

Conclusion

L'algorithme de Dijkstra est robuste mais explore souvent plus de sommets que nécessaire, ce qui le rend moins efficace pour les grands graphes. L'algorithme A*, grâce à une heuristique bien choisie (comme l'Euclidienne dans notre cas), optimise l'exploration et offre une performance supérieure dans les scénarios adaptés.

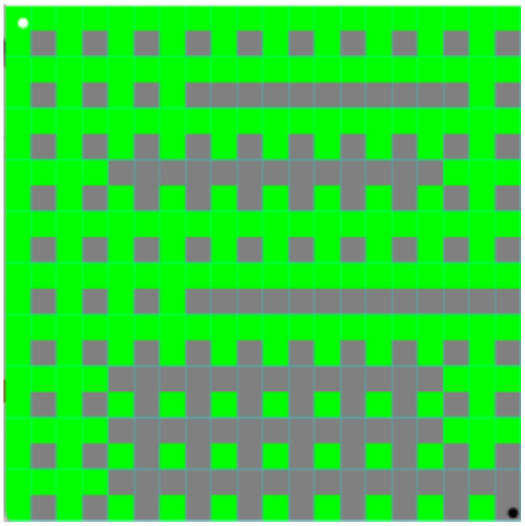

FIGURE 5 – Graphe 2

```
Done! Using Dijkstra:
    Number of nodes explored: 382
    Total time of the path: 2451.154687870049
```

FIGURE 6 – Résultats d'exécution avec Dijkstra

```
Done! Using A* (Euclide):
    Number of nodes explored: 377
    Total time of the path: 2451.154687870049
```

FIGURE 7 – Résultats d'exécution avec A*

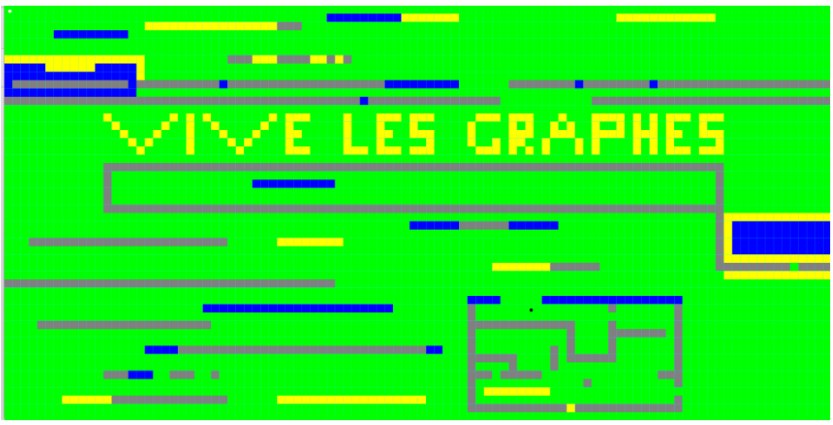

FIGURE 8 – Graphe 3

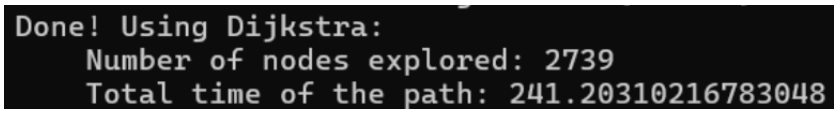

FIGURE 9 – Résultats d'exécution avec Dijkstra

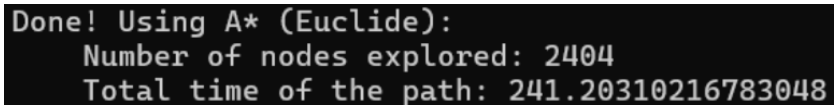

FIGURE 10 – Résultats d'exécution avec A*