RECHERCHE DE CHEMIN PAR A*

KANT ALIAGAS Cassandre

CMI-INFO 2023

INTRODUCTION

L'algorithme A* est l'un des algorithmes les plus populaires et efficaces utilisés dans le domaine de l'intelligence artificielle et de la recherche de chemins. Il a été développé à l'origine par Peter Hart, Nils Nilsson et Bertram Raphael en 1968, et depuis lors, il est devenu un pilier essentiel de nombreux systèmes de planification et de navigation automatisés.

L'objectif fondamental de l'algorithme A* est de trouver l'un des plus courts chemins entre un nœud de départ et un nœud d'arrivée dans un graphe pondéré.

Cela en fait un outil précieux pour de nombreux problèmes de recherche de chemins, tels que la planification de trajets dans les systèmes de navigation GPS, la recherche de chemins optimaux pour les robots mobiles, les jeux vidéo lors des déplacement d'un personnage non joueur et bien d'autres applications.

SOMMAIRE

Dans ce rapport, nous examinerons en détails l'algorithme A* sous différent angles techniques qui sont les suivants :

- **Les termes techniques servant à comprendre l'algorithme**
- **La théorie de la recherche**
- **Le déroulement du programme**
- **Les avantages de A***
- **Les limites de A***
- **Le comparatifs avec Dijkstra**

Vous trouverez en annexe, le code de A* afin de trouver l'un des chemin les plus courts dans un graph pondéré.

LES TERMES TECHNIQUES

Définition et Vocabulaire

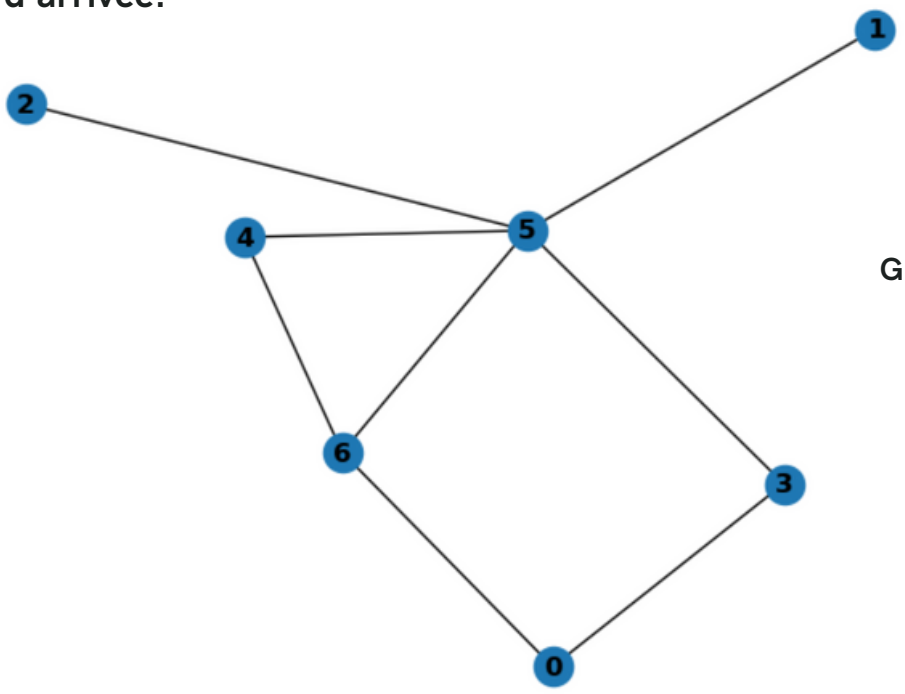
Graphe : **Comme vous l'avez vu ci-dessous un graphe contient des nœud et des arrêtes. Il peut être orienté lorsque le sens est fixe (On pourrait passer du point A à B mais pas du point B à A) mais dans notre cas il sera non orienté.**

Nœud : **Ils sont en bleu sur l'exemple ci-dessus. Ils sont les intersections entre les différentes arrêtes.**

Distance : **La distance est celle qu'on a entre deux nœuds.**

Cout : **Il est composé des distances parcourus plus le nœud futur.**

Heuristique : **Il nous sert à orienter notre recherche, il est égal à la distance à vol d'oiseau entre le nœud actuellement parcourus et le nœud d'arrivée.**



Graph comprenant 5 nœud

LE DÉROULEMENT DU PROGRAMME

Le fonctionnement théorique de l'algorithme A* repose sur une recherche guidée par une estimation heuristique du coût restant pour atteindre l'objectif. Voici les étapes principales de l'algorithme :

Initialisation : L'algorithme A* débute en initialisant une liste de nœuds à explorer. Le nœud de départ est ajouté à cette liste, et ses attributs tels que le coût total (g), le coût heuristique estimé (h) et le coût total estimé (f = g + h) sont calculés.

Sélection du nœud : À chaque itération, le nœud avec le coût total estimé le plus faible est extrait de la liste des nœuds à explorer. Ce nœud devient le nœud courant.

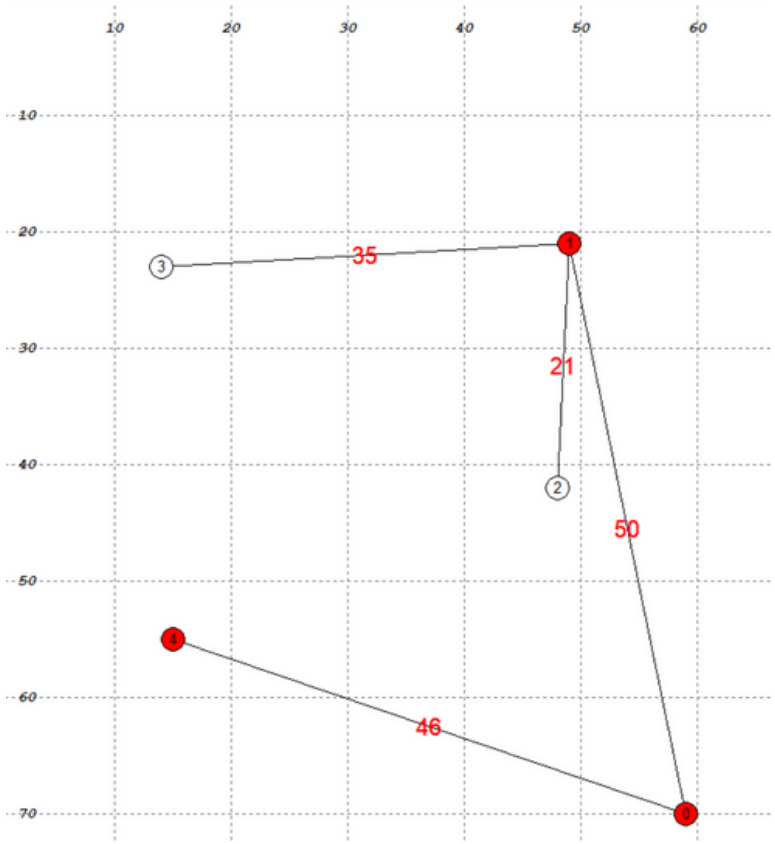
Vérification de l'objectif : Le nœud courant est vérifié pour déterminer s'il correspond à l'objectif recherché. Si c'est le cas, l'algorithme a trouvé un chemin optimal jusqu'à l'objectif, et la recherche se termine.

Expansion des nœuds : Si le nœud courant n'est pas l'objectif, il est alors étendu en générant tous ses successeurs possibles. Ces successeurs sont les nœuds voisins accessibles à partir du nœud courant.

Mise à jour des coûts : Pour chaque successeur, l'algorithme calcule les coûts associés. Le coût de déplacement depuis le nœud de départ jusqu'au successeur (g) est mis à jour en ajoutant le coût de déplacement entre le nœud courant et le successeur. Le coût heuristique estimé (h) est également calculé pour estimer le coût restant pour atteindre l'objectif à partir du successeur. Le coût total estimé (f = g + h) est ensuite mis à jour.

Ajout des nœuds à explorer : Les successeurs sont ajoutés à la liste des nœuds à explorer, à condition qu'ils ne soient pas déjà présents dans la liste ou dans une autre liste de nœuds explorés. Si un successeur est déjà présent, l'algorithme compare les coûts pour déterminer s'il est préférable de mettre à jour le nœud existant avec de nouveaux coûts.

Répétition : Les étapes 2 à 6 sont répétées jusqu'à ce que l'objectif soit atteint ou que la liste des nœuds à explorer soit vide. Si la liste est vide et que l'objectif n'a pas été atteint, cela signifie qu'aucun chemin n'existe.
Reconstruction du chemin : Si l'objectif est atteint, un chemin optimal est reconstruit en remontant les nœuds parents depuis l'objectif jusqu'au nœud de départ.



On demande au programme d'aller du nœud 1 au nœud 4. Il va explorer les noeuds 1,2,0,4 mais il retrounera le chemin 1,0,4.

Canvas retourné par tkinter (librairie python cf. annexe)

AVANTAGES DE A*

L'algorithme A* présente plusieurs avantages majeurs qui en font l'un des algorithmes les plus largement utilisés dans la recherche de chemins et la planification :

Complétude : L'algorithme A* garantit de trouver une solution si elle existe, à condition que la recherche soit effectuée dans un espace de recherche fini et qu'il n'y ait pas d'erreurs dans les données d'entrée. Il ne laisse aucun chemin inexploré tant que l'objectif n'a pas été atteint ou qu'il n'y a plus de nœuds à explorer.

Optimalité : Lorsque l'heuristique utilisée est admissible (elle ne surestime pas le coût restant), A* garantit de trouver un chemin optimal, c'est-à-dire le chemin avec le coût total le plus bas parmi tous les chemins possibles. Cela en fait un choix idéal lorsque la recherche de la solution la plus économique est cruciale.

Efficacité : Grâce à l'estimation heuristique, A* peut guider la recherche vers les chemins les plus prometteurs, évitant ainsi d'explorer inutilement des zones peu pertinentes de l'espace de recherche. Cela réduit considérablement le nombre de nœuds à explorer par rapport à des algorithmes comme la recherche en largeur, rendant A* plus rapide et plus efficace pour trouver des solutions.

En résumé, l'algorithme A* offre la garantie de trouver une solution optimale, tout en étant efficace et adaptable. Sa combinaison de recherche en largeur et de recherche en profondeur ciblée grâce à une heuristique en fait un outil puissant pour résoudre une variété de problèmes de recherche de chemins, contribuant ainsi à des applications pratiques et à des avancées technologiques significatives.

LIMITES DE A*

Bien que l'algorithme A* présente de nombreux avantages, il présente également certaines limites qu'il est important de prendre en compte :

Heuristique non admissible : Si l'heuristique utilisée dans A* surestime le coût restant pour atteindre l'objectif, cela peut entraîner une recherche inefficace et la possibilité de ne pas trouver la solution optimale. Dans certains cas, cela peut également conduire à l'exploration de chemins non optimaux.

Sensibilité à la topologie de l'espace de recherche : L'efficacité de l'algorithme A* peut être affectée par la structure de l'espace de recherche. Dans certains cas, des configurations spécifiques peuvent conduire à des résultats sous-optimaux ou à des temps de calcul plus longs, ce qui rend l'algorithme moins performant.

Dépendance de la fonction heuristique : La qualité et la pertinence de la fonction heuristique utilisée dans A* peuvent grandement influencer les performances de l'algorithme. Trouver une heuristique appropriée peut être un défi dans certains problèmes complexes, et une mauvaise estimation peut entraîner des résultats peu fiables ou une exploration inefficace de l'espace de recherche.

Il est essentiel de prendre en compte ces limites lors de l'utilisation de l'algorithme A* et de les évaluer en fonction du contexte spécifique du problème. Dans certains cas, des variantes ou des approches alternatives peuvent être plus appropriées pour surmonter ces limitations et répondre aux exigences particulières du problème à résoudre.

COMPARATIF A* DIJKSTRA

L'algorithme de Dijkstra et l'algorithme A* sont deux algorithmes couramment utilisés dans la recherche de chemins, mais ils diffèrent dans leur approche et leurs performances. Voici une comparaison entre les deux :

Objectif : L'algorithme de Dijkstra est conçu pour trouver le chemin le plus court entre un nœud de départ et tous les autres nœuds d'un graphe. En revanche, l'algorithme A* est conçu pour trouve l'un des plus courts chemins.

Heuristique : L'algorithme de Dijkstra n'utilise pas d'heuristique. Il évalue les chemins en fonction de leurs coûts réels accumulés depuis le nœud de départ. En revanche, A* utilise une heuristique pour estimer le coût restant jusqu'à l'objectif. Cela lui permet d'explorer en priorité les chemins les plus prometteurs, ce qui peut conduire à une recherche plus efficace et à une meilleure performance en termes de temps de calcul.

Complexité : En termes de complexité, l'algorithme de Dijkstra a une complexité temporelle de (|V| + |E|)*log |V|, où |V| est le nombre de nœuds et |E| est le nombre d'arêtes dans le graphe. En revanche, la complexité temporelle de A* dépend de l'efficacité de l'heuristique utilisée. Dans le pire des cas, la complexité de A* est similaire à celle de Dijkstra, mais dans les meilleurs cas où l'heuristique est bien choisie, A* peut être beaucoup plus rapide.

Optimisation : L'algorithme de Dijkstra garantit de trouver le chemin le plus court entre le nœud de départ et tous les autres nœuds, mais il n'est pas spécifiquement optimisé pour trouver le chemin le plus court vers un nœud d'arrivée spécifique. En revanche, l'algorithme A* est conçu pour trouver un chemin optimal vers un nœud d'arrivée spécifique, à condition que l'heuristique soit admissible.

Domaines d'application : L'algorithme de Dijkstra est largement utilisé dans les problèmes de routage de réseaux, les problèmes de plus court chemin dans les graphes non dirigés, et la recherche de chemins sans contraintes particulières. A* est particulièrement adapté aux problèmes de recherche de chemins dans des domaines tels que la robotique, les jeux vidéo, la planification de trajets, où la recherche d'un chemin optimal vers un objectif spécifique est cruciale.

En résumé, l'algorithme de Dijkstra est plus adapté aux problèmes de recherche de chemins sans objectif spécifique, tandis que l'algorithme A* est optimisé pour trouver un chemin optimal vers un nœud d'arrivée spécifique grâce à l'utilisation d'une heuristique. Le choix entre les deux dépendra des exigences spécifiques du problème et des contraintes du domaine d'application.

ANNEXE

Voici un lien pour accéder à la page où se trouve le code écrit en python :