Département de Génie Informatique

INF444 - Intelligence Artificielle
Projet

Etudiants

Zeynep Selin	Uygun	19401824
Berkay	Bayazıt	20401900
Mustafa Mert	Yelken	19401821

14 janvier 2024

Encadtants

Dr.Öğr.Üyesi Pınar Uluer

Table des matières

1 Description de l'Algorithme A*

L'algorithme A* est une méthode de recherche par chemin avancée qui combine les avantages de la recherche en largeur (BFS) et la recherche de chemin le plus court (comme l'algorithme de Dijkstra). Il est largement utilisé dans les problèmes de parcours de graphe, notamment dans les jeux, la robotique et la planification d'itinéraires.

1.1 Principe de Fonctionnement

L'algorithme A* recherche le chemin le plus court entre un état initial et un état final en minimisant une fonction d'évaluation 'f(n) = g(n) + h(n)', où :

- 'n' représente un nœud (ou un état) du graphe (dans le cas du puzzle 8, une disposition particulière des tuiles).
- 'g(n)' est le coût du chemin de l'état initial à 'n'.
- 'h(n)' est une heuristique qui estime le coût le moins cher de 'n' à l'état final.

1.2 Heuristique

L'efficacité de l'algorithme A* dépend fortement de l'heuristique utilisée. Pour le puzzle 8, une heuristique courante est la distance de Manhattan, qui calcule la somme des distances absolues des positions actuelles des tuiles par rapport à leurs positions cibles.

1.3 Exploration des États

- L'algorithme maintient deux ensembles : un ensemble ouvert (les nœuds à explorer) et un ensemble fermé (les nœuds déjà explorés).
- Au début, seul le nœud initial est dans l'ensemble ouvert.
- À chaque étape, l'algorithme retire le nœud avec la valeur de 'f(n)' la plus basse de l'ensemble ouvert, l'explore, puis l'ajoute à l'ensemble fermé.
- Les voisins de ce nœud sont ajoutés à l'ensemble ouvert s'ils n'ont pas été explorés ou si un meilleur chemin vers eux est trouvé.
- Le processus se poursuit jusqu'à ce que l'état final soit atteint ou que l'ensemble ouvert soit vide.

2 Raisons du Choix de l'Algorithme A*

Nous avons opté pour l'implémentation de l'algorithme A* pour notre projet en raison de ses multiples avantages qui le rendent particulièrement adapté à notre cas d'utilisation. Les principales raisons de ce choix sont :

Efficacité : L'algorithme A* surpasse les méthodes de recherche naïve. Grâce à l'utilisation d'une fonction heuristique, il oriente la recherche de manière plus ciblée, réduisant

ainsi le nombre de chemins explorés inutilement.

Heuristique Informée : L'emploi de l'heuristique de la distance de Manhattan permet des estimations précises du nombre de mouvements restants pour atteindre le but, accélérant la recherche vers la solution.

Optimalité et Complétude : Avec une heuristique admissible, c'est-à-dire ne surestimant jamais le coût réel pour atteindre l'objectif, l'algorithme A* garantit de trouver la solution optimale si elle existe.

Flexibilité : L'algorithme peut être aisément adapté à différentes variantes du puzzle 8 en modifiant simplement la fonction heuristique.

3 Avantages et Inconvénients de l'Algorithme A*

L'algorithme A* présente plusieurs avantages et inconvénients qui influencent son application dans divers contextes de résolution de problèmes. Voici un aperçu détaillé de ces points.

3.1 Avantages

- **Optimalité :** L'algorithme A* garantit de trouver la solution la plus courte si l'heuristique employée est admissible, c'est-à-dire qu'elle ne surestime jamais le coût réel pour atteindre l'objectif.
- **Efficace en Temps :** A* est plus rapide que les méthodes de recherche non informées telles que la recherche en largeur (BFS), grâce à la guidance fournie par l'heuristique qui permet de se diriger plus directement vers la solution.
- **Espace de Recherche Réduit :** L'algorithme explore moins d'états que d'autres méthodes plus naïves, ce qui se traduit par une économie de mémoire significative.

3.2 Inconvénients

- **Complexité Spatiale :** A* peut consommer beaucoup de mémoire, particulièrement dans les cas où le puzzle ou le problème à résoudre possède de nombreux états possibles. Cette consommation mémoire est due à la nécessité de stocker les états dans l'ensemble ouvert et fermé.
- **Dépendance à l'Heuristique :** La performance de l'algorithme A* dépend fortement de la qualité de l'heuristique utilisée. Une heuristique mal choisie ou inappropriée peut entraîner de mauvaises performances et une exploration inefficace de l'espace d'états.
- **Calcul de l'Heuristique :** Chaque étape de l'algorithme nécessite le calcul de la fonction heuristique pour chaque nœud, ce qui peut s'ajouter à la charge de

calcul globale. Cette exigence peut rendre l'algorithme moins efficace, en particulier lorsque le calcul de l'heuristique est complexe ou coûteux en termes de temps de traitement.

4 Rapport sur les Performances des Algorithmes pour le Puzzle 8

Ce rapport offre une analyse comparative des performances de trois algorithmes - A*, BFS (Recherche en Largeur) et IDDFS (Recherche en Profondeur Itérative) - lorsqu'ils sont appliqués au Puzzle 8. Nous évaluons la complétude, l'optimalité, ainsi que les complexités temporelle et spatiale, et rapportons également les résultats obtenus lors de nos tests.

4.1 Résultats des Tests

Les temps de résolution pour 15 configurations différentes du puzzle ont été mesurés pour chaque algorithme. Les moyennes des temps de résolution (en nanosecondes) sont les suivantes :

- A* : 12,897,280 ns
- BFS : 11,706,637,630 ns
- IDDFS : 218,113,910 ns

4.2 Analyse Comparative

4.2.1 Complétude

- **A*** : Complet si l'heuristique est admissible (ne surestime pas le coût réel).
- **BFS** : Complet mais peut être très lent et nécessiter beaucoup de mémoire.
- **IDDFS** : Complet et souvent plus rapide que BFS, en particulier dans des espaces de recherche profonds.

4.2.2 Optimalité

- **A*** : Optimal si l'heuristique est admissible.
- **BFS** : Optimal car il explore systématiquement tous les chemins.
- **IDDFS** : Non optimal, car il peut trouver des solutions plus longues avant les plus courtes.

4.2.3 Complexité Temporelle

- **A*** : Théoriquement, la complexité temporelle de A* est $O(b^d)$, où b est le facteur de branchement et d la profondeur de la solution la moins coûteuse. Cette complexité varie fortement selon l'heuristique.
- **BFS** : La complexité temporelle de BFS est $O(b^d)$, ce qui la rend impraticable pour des espaces de recherche très vastes.

— **IDDFS** : La complexité temporelle est également $O(b^d)$, mais avec une utilisation de la mémoire plus efficace.

4.2.4 Complexité Spatiale

— **A*** : La complexité spatiale est $O(b^d)$, potentiellement problématique pour des espaces de recherche vastes.
— **BFS** : Également $O(b^d)$, BFS est très gourmand en mémoire pour des problèmes à grande échelle.
— **IDDFS** : Avec une complexité spatiale en $O(bd)$, IDDFS est nettement plus économique en mémoire que BFS.

5 Notre meilleur score

Nous avons obtenu un meilleur score de 0.547 seconds. Étant donné que l'algorithme fonctionne très rapidement, presque tout ce temps est le temps que nous passons jusqu'à ce que nous appuyions sur le bouton « Solve Puzzle ».