

Projet d'Informatique Scientifique: Path Finding

Cette présentation offre un aperçu structuré de mon projet d'informatique scientifique. Elle aborde les aspects essentiels, de la motivation initiale aux perspectives futures.



par Karl Philippe Jean

Introduction et Contexte du Projet

Contexte

La recherche de chemin est un problème central en robotique, planification de trajets, IA et jeux vidéo. Différents algorithmes sont employés selon les contraintes : optimalité ou rapidité. Ce projet explore des approches classiques (**Dijkstra**, **A***, **BFS**) et des variantes (**WA***, recherche **gloutonne**), comparant leurs performances et applications.

Importance

L'étude de ces algorithmes révèle leurs atouts et faiblesses :

- **Dijkstra** : chemin le plus court garanti, mais coûteux en temps.
- **A*** : combine heuristique et optimisation, plus efficace.
- **WA*** : compromis optimalité/rapidité.
- **BFS** : utile pour graphes non pondérés.
- **Recherche gloutonne** : explore les chemins prometteurs, sans garantie d'optimalité.

L'implémentation permet une évaluation pratique de ces compromis.

Problématique et Objectifs

Problématique

La recherche de chemin est un enjeu majeur en informatique. Le choix d'un algorithme (Dijkstra, A*, etc.) dépend du graphe et des contraintes (temps, optimalité). Il faut arbitrer entre précision et efficacité.

Questions clés :

1. **Quand** privilégier un algorithme ?
2. **Comment** l'optimiser ?
3. **Quel compromis** adopter (optimalité/temps) ?

Objectifs

Ce projet compare des algorithmes de recherche de chemin afin de :

1. **Évaluer l'efficacité** (Dijkstra, A*, WA*, BFS, gloutonne) selon :
 - o **Temps d'exécution.**
 - o **Mémoire utilisée.**
 - o **Nombre d'états explorés.**
 - o **Optimalité du chemin.**
2. **Déterminer l'influence** du graphe (poids, structure, taille).
3. **Optimiser les algorithmes** (heuristiques, paramètres) pour un bon compromis.
4. **Fournir des recommandations** pour le choix d'algorithme selon l'application (jeux, robotique, etc.).
5. **Développer une implémentation** modulaire pour tester/adapter des algorithmes.

Méthodes Numériques

Comparaison expérimentale des algorithmes.

1. Modélisation du Graphe :

Graph = carte (nœuds + coûts).

2. Implémentation des Algorithmes :

Structures adaptées (tas binaire, file d'attente).

Calcul du chemin.

1. Évaluation des Performances :

Basée sur :

Tests sur cartes variées.

- Temps d'exécution.
- Nombre d'états explorés.
- Optimalité.
- Mémoire utilisée.

1. Comparaison des Algorithmes :

Analyse comparative.

2. Visualisation des Résultats :

Graphiques comparatifs.



Modélisation

• Optimisation (A^* et WA^*) :

Heuristique, pondération.

• Recherche Exhaustive :

Dijkstra = optimal, BFS = rapide.

• Recherche Gloutonne :

Nœud prometteur, rapide (non optimal).

• Modularité et Flexibilité :

Code modulaire.

• Cas Réels et Synthétiques :

Cartes variées.

• Tests de Scalabilité :

Graphes de taille croissante.

2

Résultats Principaux

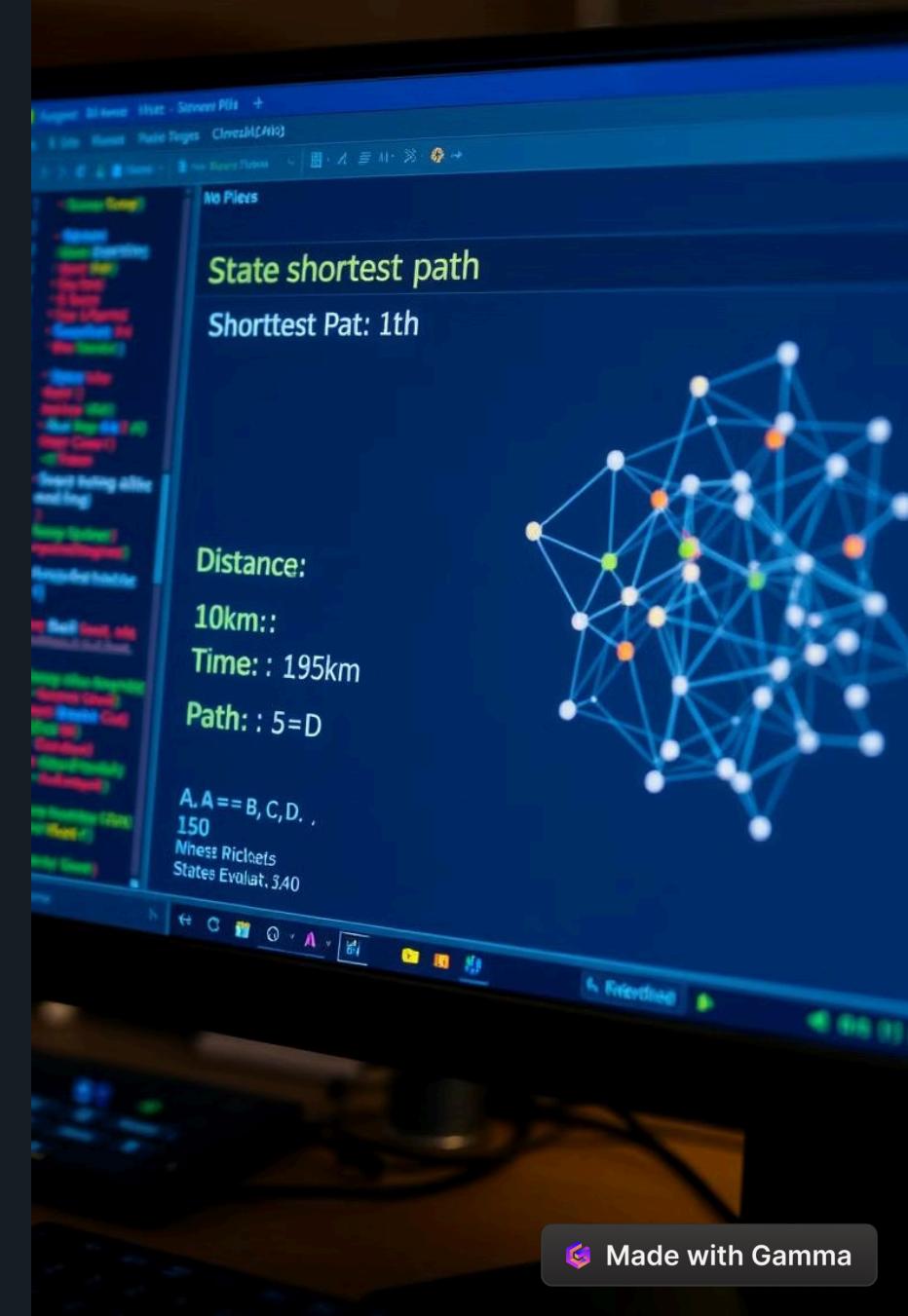
⌚ Temps d'exécution

➡ Distance

👁 Chemin parcouru

✗ États évalués

Illustrations des résultats avec analyse quantitative et comparaison avec la littérature.



Breadth-First-Search (Flood Fill)

- Trouve **le chemin le plus court en nombre de sauts** dans un graphe **non pondéré**.
- Utilise une **file FIFO** pour garantir que les sommets sont explorés par ordre de distance croissante.
- Ne prend pas en compte les coûts, considère seulement la distance en nombre de cases.
- Complexité: $O(n + m)$

```
Breadth_First_Search  
Solution :  
CPUtime (s) : 0.625121  
Distance (189, 193) → (226, 437) : 281.0  
Number of states evaluated : 127194  
Path (189, 193) → (226, 437)
```

Dijkstra

1. Garantit le chemin le plus court.
2. La file de priorité garantit que l'algorithme explore toujours le nœud avec la plus petite distance connue de la source.
3. Complexité: $O((n + m) \log n)$.

```
julia> include("path_finding.jl")
* _____DIJKSTRA_____
Solution :
CPUtime (s) : 0.469048
Distance (189, 193) → (226, 437) : 335.0
Number of states evaluated : 91251
Path (189, 193) → (226, 437)
```



A*

- Garantit le chemin le plus court si l'heuristique est admissible (sous-estime toujours le coût réel).
- Utilise une fonction heuristique pour prioriser les nœuds à explorer.
- Complexité : $O((n+m)\log n)$ dans le pire des cas.
- Il utilise une heuristique pour estimer le coût restant jusqu'à la destination.
- A* combine l'efficacité de Dijkstra avec la directionnalité de la recherche gloutonne.

```
----- A* Algorithm -----  
  
Solution :  
CPU time (s) : 0.146476  
Distance (189, 193) → (226, 437) : 335.0  
Number of states evaluated : 10854  
Path (189, 193) → (226, 437)
```

WA*

- Trouve un chemin sous-optimal, mais plus rapidement que A*.
- Un facteur w pondère l'heuristique pour accélérer la recherche au détriment de l'optimalité.
- Complexité : $O((n+m)\log n)$.

```
----- Bounded Suboptimal Search (BSS) -----
Solution :
CPU time (s) : 0.179979
Distance (189, 193) → (226, 437) : 335.0
Number of states evaluated : 4884
Path (189, 193) → (226, 437)
```



Greedy Best-First Search

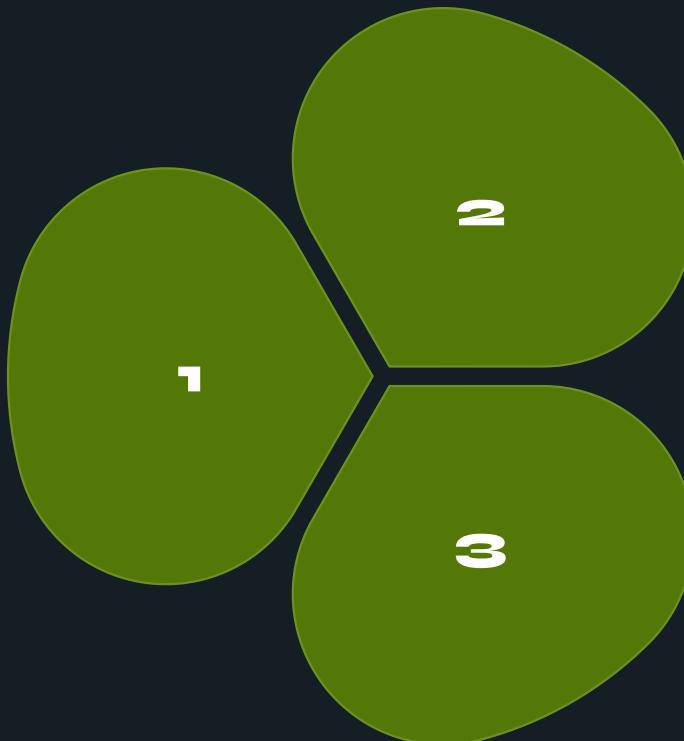
- Ne garantit pas le chemin le plus court, mais explore rapidement vers la cible.
- Se base uniquement sur l'heuristique, sans tenir compte des coûts déjà accumulés.
- **Complexité** : $O(n+m)$.

```
----- Greedy Best-First Search -----
Solution :
CPU time (s) : 0.113
Distance (189, 193) → (226, 437) : 709.0
Number of states evaluated : 282
Path (189, 193) → (226, 437)
```

Discussion et Interprétation

Contexte

Ce projet analyse plusieurs algorithmes de recherche de chemin afin d'évaluer leur efficacité en termes de rapidité et d'optimalité sur différents types de graphes.



Analyse

Les résultats révèlent que Dijkstra est optimal mais lent, A* et WA* sont efficaces grâce à l'heuristique, tandis que BFS et la recherche gloutonne offrent des alternatives plus rapides mais parfois sous-optimales

Évaluation

La comparaison des algorithmes met en évidence leurs forces et limites selon la structure du graphe, permettant de choisir la méthode la plus adaptée à chaque situation.



Perspectives et Travaux Futurs



Optimisation de l'algorithme pour les architectures parallèles et application à des problèmes industriels concrets.



Conclusion



Récapitulation

Le projet a implémenté plusieurs algorithmes de recherche de chemin (Dijkstra, A*, WA*, BFS, et recherche gloutonne) pour explorer des graphes et trouver les chemins optimaux.



Importance

Les algorithmes de recherche de chemin, jouent un rôle clé dans divers domaines comme la navigation, les jeux vidéo et les réseaux de transport. Ils optimisent la recherche d'itinéraires.

