Université de Paris

Algorithmique Avancée

Recherche du Plus Court Chemin

*Implémentation et Comparaison des Algorithmes de Dijkstra et A**

Compte Rendu Partie B

Auteur :
Ali Gouarab

Année universitaire 2024–2025

Sommaire

1 Introduction

1.1 Contexte

La recherche du plus court chemin dans un graphe est un problème fondamental en informatique, avec des applications dans de nombreux domaines, navigation GPS, routage réseau, jeux vidéo, robotique, etc. Ce projet s'inscrit dans le cadre du cours d'Algorithmique Avancée et vise à implémenter et comparer deux algorithmes classiques, l'algorithme de Dijkstra et l'algorithme A*.

1.2 Objectifs

Les objectifs de ce projet sont :

1. Implémenter l'algorithme de Dijkstra pour la recherche du plus court chemin
2. Implémenter l'algorithme A* avec plusieurs heuristiques
3. Comparer les performances des deux algorithmes sur différentes cartes
4. Analyser les cas d'utilisation réels de chaque algorithme

1.3 Organisation du rapport

Ce rapport est organisé comme suit : la section 2 présente les fondements théoriques des algorithmes. La section 3 détaille l'implémentation réalisée. La section 4 présente les résultats expérimentaux et leur analyse. Enfin, la section 5 conclut ce rapport.

2 Fondements Théoriques

2.1 Modélisation du problème

2.1.1 Représentation par graphe pondéré

Définition (Graphe pondéré) : Un graphe pondéré est un triplet $G = (V, E, w)$ où :
- V est l'ensemble des sommets (vertices)
- $E \subseteq V \times V$ est l'ensemble des arêtes (edges)
- $w : E \to \mathbb{R}^+$ est la fonction de poids

Dans notre cas, la carte 2D est modélisée comme suit :
- Chaque case de la grille correspond à un sommet
- Chaque case est connectée à ses 8 voisins (8 connexité)
- Le poids d'une arête dépend du type de terrain traversé

2.1.2 Calcul des poids

Pour une arête reliant une case A à une case B, le poids est calculé selon la formule :

$$w(A, B) = \frac{t_A + t_B}{2} \cdot d(A, B)$$

où t_X est le temps de traversée de la case X et $d(A, B)$ est la distance euclidienne :

$$d(A, B) = \begin{cases} 1 & \text{si adjacents horizontalement ou verticalement} \\ \sqrt{2} & \text{si adjacents en diagonale} \end{cases}$$

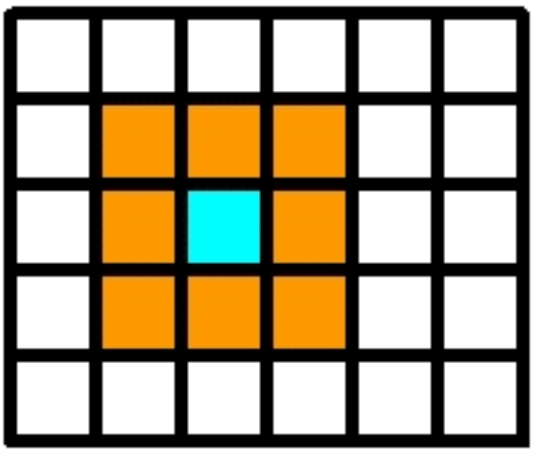

Fig. 1. – Connexité 8-voisins. Les arêtes directes ont un facteur de distance 1, les diagonales $\sqrt{2}$.

2.2 Algorithme de Dijkstra

2.2.1 Principe

L'algorithme de Dijkstra, proposé par Edsger Dijkstra en 1959, est un algorithme glouton qui trouve le plus court chemin depuis un sommet source vers tous les autres sommets d'un graphe à poids positifs.

Théorème : L'algorithme de Dijkstra calcule correctement les plus courts chemins dans un graphe à poids positifs.

2.2.2 Pseudo-code

Algorithme : Dijkstra

Entrées : Graphe $G = (V, E, w)$, sommet source s, sommet destination t
Sortie : Plus court chemin de s à t

1. $\text{dist}[v] \leftarrow \infty$ pour tout $v \in V$
2. $\text{dist}[s] \leftarrow 0$
3. $\text{prev}[v] \leftarrow \text{null}$ pour tout $v \in V$
4. $Q \leftarrow V$
5. **Tant que** $t \in Q$:

   Sélectionner $u \leftarrow \arg\min_{v \in Q} \text{dist}[v]$
   $Q \leftarrow Q \setminus \{u\}$
   Pour chaque voisin v de u :
      $\text{alt} \leftarrow \text{dist}[u] + w(u, v)$
      Si $\text{alt} < \text{dist}[v]$: mettre à jour
6. **Retourner** chemin via prev

Fig. 2. – Algorithme de Dijkstra

2.2.3 Complexité

Proposition (Complexité de Dijkstra) : Avec notre implémentation utilisant un `HashSet` :

- Complexité temporelle : $\mathcal{O}(V^2)$
- Complexité spatiale : $\mathcal{O}(V)$

Remarque : Avec une file de priorité (tas binaire), la complexité peut être réduite à $\mathcal{O}((V + E) \log V)$.

2.3 Algorithme A*

2.3.1 Principe

L'algorithme A*, développé en 1968, est une extension de Dijkstra qui utilise une heuristique pour guider la recherche vers l'objectif.

Définition (Fonction d'évaluation) : L'algorithme A* utilise une fonction d'évaluation $f(n)$ définie par :

$$f(n) = g(n) + h(n)$$

où :
- $g(n)$ est le coût réel du chemin de la source au nœud n
- $h(n)$ est l'estimation heuristique du coût de n à la destination

Définition (Heuristique admissible) : Une heuristique h est dite *admissible* si elle ne surestime jamais le coût réel :

$$\forall n \in V, \quad h(n) \leq h^*(n)$$

où $h^*(n)$ est le coût réel optimal de n à la destination.

Théorème (Optimalité de A*) : Si l'heuristique h est admissible, alors A* trouve le chemin optimal.

2.3.2 Heuristiques implémentées

Nous avons implémenté trois heuristiques classiques :

2.3.2.1 Distance Euclidienne

$$h_{\text{eucl}}(n) = \sqrt{(x_n - x_t)^2 + (y_n - y_t)^2}$$

2.3.2.2 Distance de Manhattan

$$h_{\text{manh}}(n) = |x_n - x_t| + |y_n - y_t|$$

2.3.2.3 Distance de Chebyshev

$$h_{\text{cheb}}(n) = \max(|x_n - x_t|, |y_n - y_t|)$$

Proposition (Admissibilité des heuristiques) : Dans notre contexte avec connexité 8-voisins et coût minimal de terrain égal à 1 :
- h_{cheb} est admissible (correspond au nombre minimal de pas)
- h_{eucl} est admissible (distance à vol d'oiseau)
- h_{manh} n'est **pas** admissible (peut surestimer car les diagonales sont autorisées)

2.3.3 Analyse de l'admissibilité des heuristiques

Rappelons qu'une heuristique h est admissible si et seulement si elle ne surestime jamais le coût réel optimal h^* pour atteindre la destination :

$$\forall n \in V, \quad h(n) \leq h^*(n)$$

Dans notre contexte, nous travaillons sur une grille avec connexité 8-voisins où le coût minimal d'un déplacement est :

- $c_{\text{direct}} = 1$ pour un mouvement horizontal ou vertical
- $c_{\text{diag}} = \sqrt{2}$ pour un mouvement diagonal

> Tous les terrains ont un coût $t \geq 1$. Le coût minimal d'une arête est donc 1 (horizontal/vertical) ou $\sqrt{2}$ (diagonal).

2.3.3.1 Distance Euclidienne : Admissible

Proposition : L'heuristique euclidienne $h_{\text{eucl}}(n) = \sqrt{(\Delta x)^2 + (\Delta y)^2}$ est admissible.

Démonstration :

L'heuristique euclidienne mesure la distance en ligne droite entre le nœud n et la destination t. Or, tout chemin réel dans la grille est une suite de segments de longueur ≥ 1.

Par l'inégalité triangulaire, la somme des longueurs d'un chemin polygonal est toujours supérieure ou égale à la distance en ligne droite entre ses extrémités :

$$h^*(n) \geq d_{\text{eucl}}(n, t) = h_{\text{eucl}}(n)$$

Intuitivement : on ne peut pas aller plus vite qu'en ligne droite.

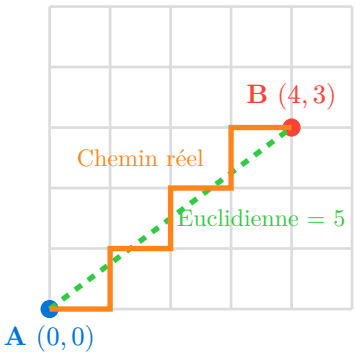

Fig. 3. – L'euclidienne (vert) est toujours inférieure ou égale au chemin réel (orange)

2.3.3.2 Distance de Chebyshev : Admissible

Proposition : L'heuristique de Chebyshev $h_{\text{cheb}}(n) = \max(|\Delta x|, |\Delta y|)$ est admissible.

Démonstration :

Soit n un nœud à la position (x_n, y_n) et t la destination à (x_t, y_t). Posons $\Delta x = |x_n - x_t|$ et $\Delta y = |y_n - y_t|$.

Le nombre minimal de pas pour atteindre t depuis n est exactement $\max(\Delta x, \Delta y)$: on effectue $\min(\Delta x, \Delta y)$ pas diagonaux (qui réduisent simultanément les deux écarts), puis $|\Delta x - \Delta y|$ pas horizontaux ou verticaux.

Le coût optimal est donc :

$$h^*(n) \geq \min(\Delta x, \Delta y) \cdot \sqrt{2} + |\Delta x - \Delta y| \cdot 1$$

Or, montrons que $h_{\text{cheb}}(n) \leq h^*(n)$:

Sans perte de généralité, supposons $\Delta x \geq \Delta y$. Alors :

$$h^*(n) \geq \Delta y \cdot \sqrt{2} + (\Delta x - \Delta y) \cdot 1 = \Delta x + \Delta y\big(\sqrt{2} - 1\big)$$

Puisque $\sqrt{2} - 1 \approx 0.414 > 0$, on a :

$$h^*(n) \geq \Delta x = \max(\Delta x, \Delta y) = h_{\text{cheb}}(n)$$

Donc Chebyshev ne surestime jamais.

Interprétation : Chebyshev compte le nombre minimal de « coups » nécessaires (comme un roi aux échecs). Chaque coup coûte au moins 1, donc l'estimation est toujours inférieure ou égale au coût réel.

2.3.3.3 Distance de Manhattan : Non admissible

Proposition : L'heuristique de Manhattan $h_{\text{manh}}(n) = |\Delta x| + |\Delta y|$ n'est **pas** admissible dans une grille 8-connexe.

Démonstration (par contre-exemple) :

Considérons $n = (0, 0)$ et $t = (1, 1)$.

L'heuristique Manhattan donne :

$$h_{\text{manh}}(n) = |1 - 0| + |1 - 0| = 2$$

Le coût réel optimal est un unique pas diagonal :

$$h^*(n) = \sqrt{2} \approx 1.414$$

On a donc :

$$h_{\text{manh}}(n) = 2 > 1.414 = h^*(n)$$

L'heuristique **surestime** le coût réel, elle n'est donc pas admissible.

Explication : Manhattan suppose qu'on ne peut se déplacer qu'horizontalement et verticalement (comme dans les rues de Manhattan). Elle ignore les raccourcis diagonaux et compte donc systématiquement un chemin plus long que le chemin réel optimal.

Généralisation : Pour tout déplacement où $\Delta x > 0$ et $\Delta y > 0$:

$$h_{\text{manh}} = \Delta x + \Delta y > \sqrt{\Delta x^2 + \Delta y^2} \geq h^*$$

La surestimation est maximale pour les déplacements purement diagonaux ($\Delta x = \Delta y$) où :

$$\frac{h_{\text{manh}}}{h^*} = \frac{2\Delta x}{\sqrt{2}\Delta x} = \sqrt{2} \approx 1.41$$

Remarque importante : Bien que Manhattan ne soit pas admissible en 8-connexité, elle reste très utilisée car :

- Elle est plus rapide à calculer
- Elle explore souvent moins de nœuds
- Dans de nombreux cas pratiques, le chemin trouvé reste proche de l'optimal

En 4-connexité (sans diagonales), Manhattan devient admissible.

2.3.4 Pseudo-code

Algorithme : A*

Entrées : Graphe G, source s, destination t, heuristique h
Sortie : Plus court chemin de s à t

1. $g[v] \leftarrow \infty$ pour tout $v \in V$
2. $g[s] \leftarrow 0$
3. prev$[v] \leftarrow$ null pour tout $v \in V$
4. $Q \leftarrow V$
5. **Tant que** $t \in Q$:

    $u \leftarrow \arg\min_{v \in Q}(g[v] + h(v))$
    $Q \leftarrow Q \setminus \{u\}$
    Pour chaque voisin v de u :
        alt $\leftarrow g[u] + w(u, v)$
        Si alt $< g[v]$:
            $g[v] \leftarrow$ alt
            prev$[v] \leftarrow u$
6. **Retourner** Chemin reconstruit via prev

Fig. 4. – Algorithme A*

3 Implémentation

3.1 Architecture du projet

Le projet est organisé selon la structure suivante :

```
Algo_Partie_B/
├── src/
│   └── up/
│       └── MainApp/
│           ├── App.java            # Classe principale
│           └── WeightedGraph.java  # Structure de données
├── maps/                           # Cartes en .txt
└── PathFinder.jar                  # Exécutable jar
```

3.2 Structure de données

3.2.1 Classe `WeightedGraph`

La structure de graphe pondéré est composée de trois classes imbriquées :

```java
public class WeightedGraph {
    static class Edge {
        int source;
        int destination;
        double weight;
    }

    static class Vertex {
        double indivTime;       // Cout du terrain
        double timeFromSource;  // Distance depuis la source
        double heuristic;       // Valeur heuristique (A*)
        Vertex prev;            // Sommet precedent
        LinkedList<Edge> adjacencylist;
        int num;
    }

    static class Graph {
        ArrayList<Vertex> vertexlist;
        int num_v;
    }
}
```

3.3 Interface en ligne de commande

L'application supporte les arguments suivants :

Option	Valeurs	Description
`-a`, `--algorithme`	`dijkstra`, `astar`	Choix de l'algorithme
`-h`, `--heuristique`	`euclidean`, `manhattan`, `chebyshev`	Heuristique pour A*
`--help`		Affiche l'aide

Tableau 1. – Arguments de la ligne de commande

```
# Exemples d'utilisation
java -jar PathFinder.jar graph.txt
java -jar PathFinder.jar -a astar graph.txt
java -jar PathFinder.jar -a astar -h manhattan graph.txt
java -jar PathFinder.jar --help
```

3.4 Format des cartes

Les cartes sont décrites dans des fichiers texte selon le format suivant :

```
==Metadata==
=Size=
nlines=50
ncol=100
=Types=
G=1
green
W=1000
gray
==Graph==
GGGGGGGGGGWWWWWGGGGG...
...
==Path==
Start=0,0
Finish=36,63
```

4 Résultats Expérimentaux

4.1 Protocole expérimental

4.1.1 Cartes de test

Nous avons utilisé plusieurs cartes pour évaluer les algorithmes :

1. **Carte standard** (`graphe.txt`) : terrain varié avec obstacles
2. **Labyrinthe** (`labyrinthe.txt`) : structure en spirale
3. **Terrain uniforme** (`uniforme.txt`) : uniquement de l'herbe

4.1.2 Métriques

Les métriques suivantes sont mesurées :

- Nombre de nœuds explorés
- Coût total du chemin trouvé

4.2 Carte standard

4.2.1 Exécution de Dijkstra

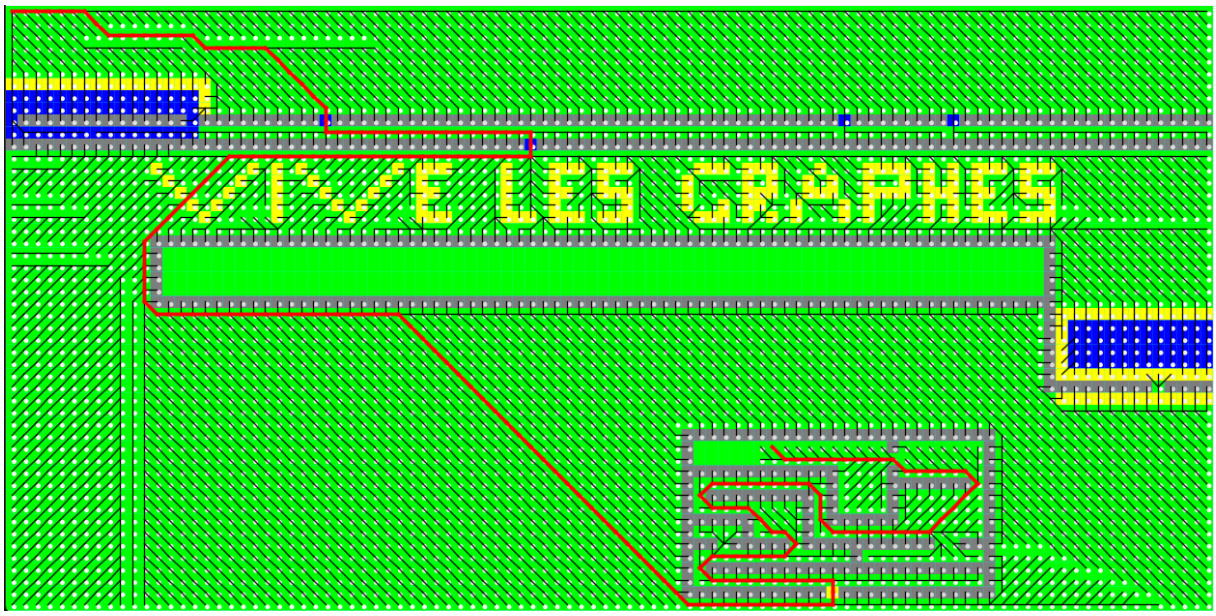

Fig. 5. – Exécution de Dijkstra sur la carte standard

4.2.2 Exécution de A*

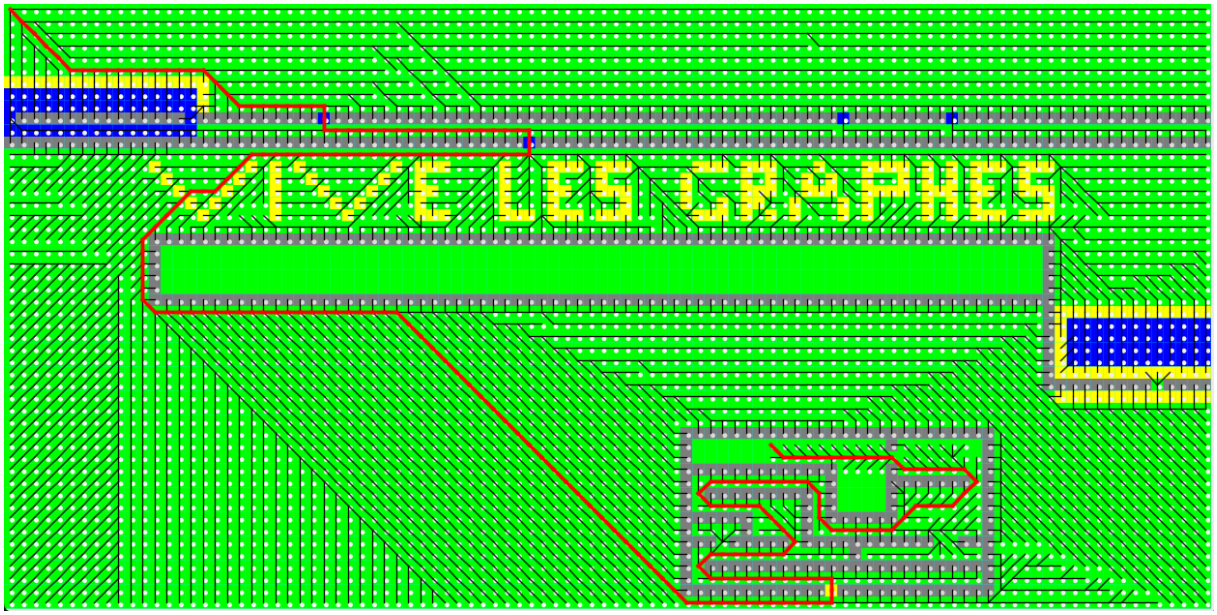

Fig. 6. – Exécution de A* (euclidien) sur la carte standard

4.2.3 Comparaison

Algorithme	Nœuds explorés	Coût du chemin	Optimal ?
Dijkstra	4156	302.6101730552669	Oui
A* (Euclidien)	4130	302.6101730552669	Oui
A* (Manhattan)	4107	303.4386001800131	Non
A* (Chebyshev)	4135	302.6101730552669	Oui

Tableau 2. – Comparaison des algorithmes sur la carte standard

Analyse : Sur cette carte, A* avec Manhattan explore le moins de nœuds (4107) mais trouve un chemin sous-optimal (303.44 > 302.61). Les heuristiques admissibles (Euclidien, Chebyshev) trouvent toutes le chemin optimal.

4.3 Carte uniforme

La carte uniforme (uniquement de l'herbe, coût = 1 partout) permet d'observer le comportement des algorithmes sans influence du terrain.

4.3.1 Exécution de Dijkstra

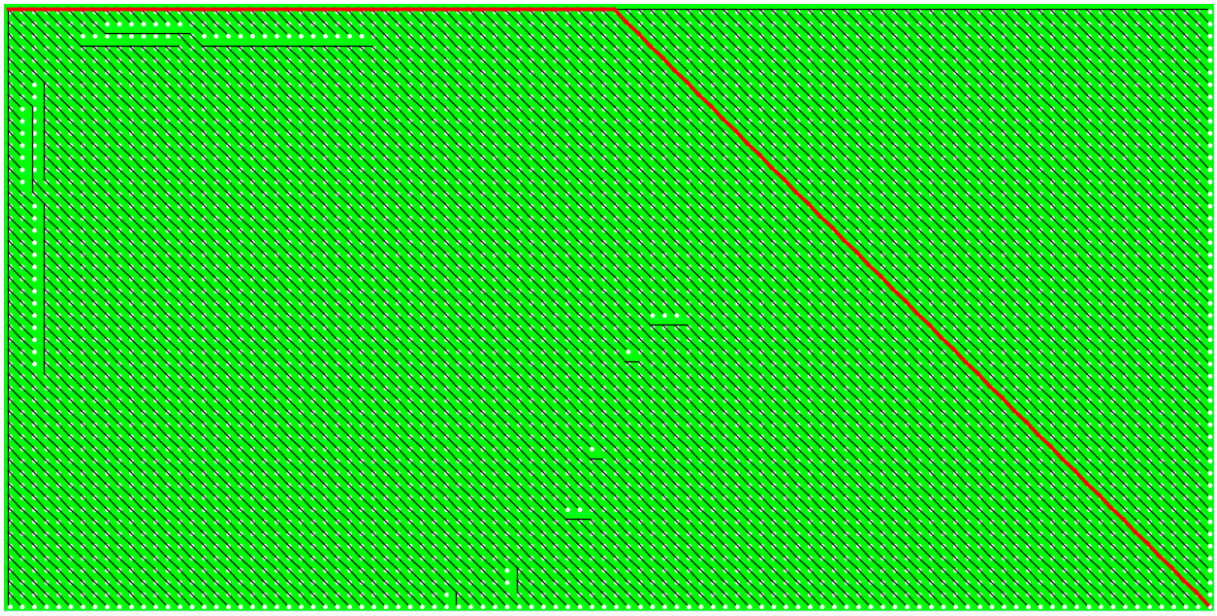

Fig. 7. – Exécution de Dijkstra sur la carte uniforme

4.3.2 Exécution de A*

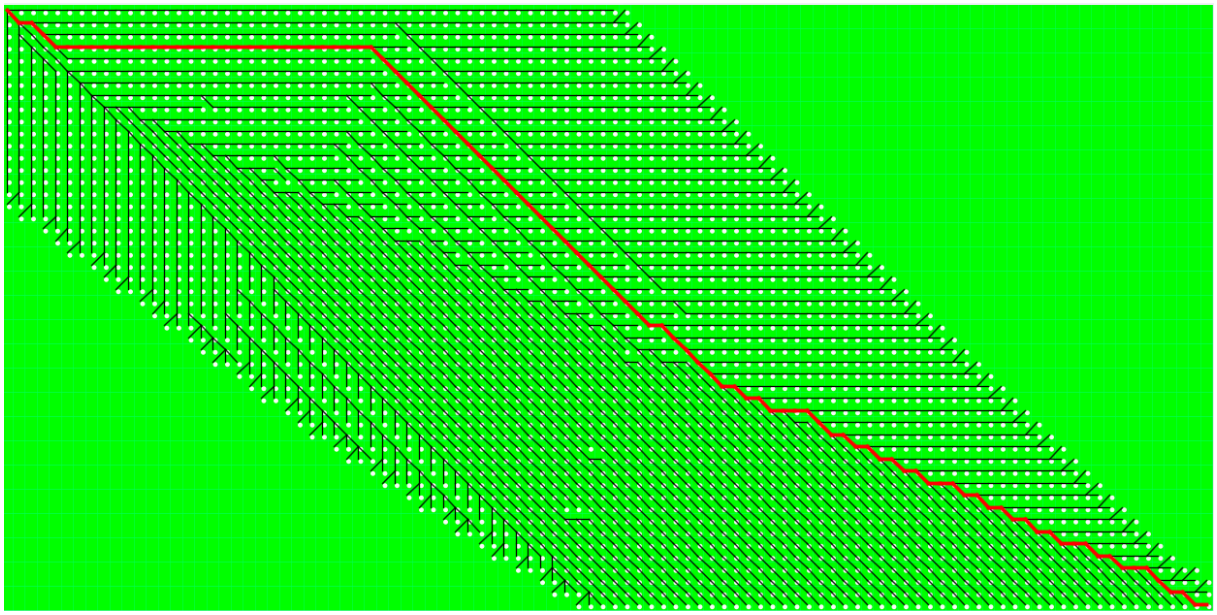

Fig. 8. – Exécution de A* (euclidien) sur la carte uniforme

4.3.3 Comparaison

Algorithme	Nœuds explorés	Coût du chemin	Optimal ?
Dijkstra	5000	119.29646455628152	Oui
A* (Euclidien)	2844	119.29646455628152	Oui
A* (Manhattan)	100	119.2964645562816	Non
A* (Chebyshev)	3484	119.29646455628152	Oui

Tableau 3. – Comparaison des algorithmes sur la carte uniforme

Analyse : Sur terrain uniforme, la différence entre Dijkstra et A* est maximale car l'heuristique guide efficacement vers la destination sans être perturbée par des variations de coût de terrain. On observe typiquement une réduction significative du nombre de nœuds explorés avec A*.

4.4 Labyrinthe

Le labyrinthe est particulièrement intéressant car le chemin direct est bloqué par des murs, forçant un long détour. L'heuristique « voit » la destination proche mais le chemin réel est très long.

4.4.1 Exécution de Dijkstra

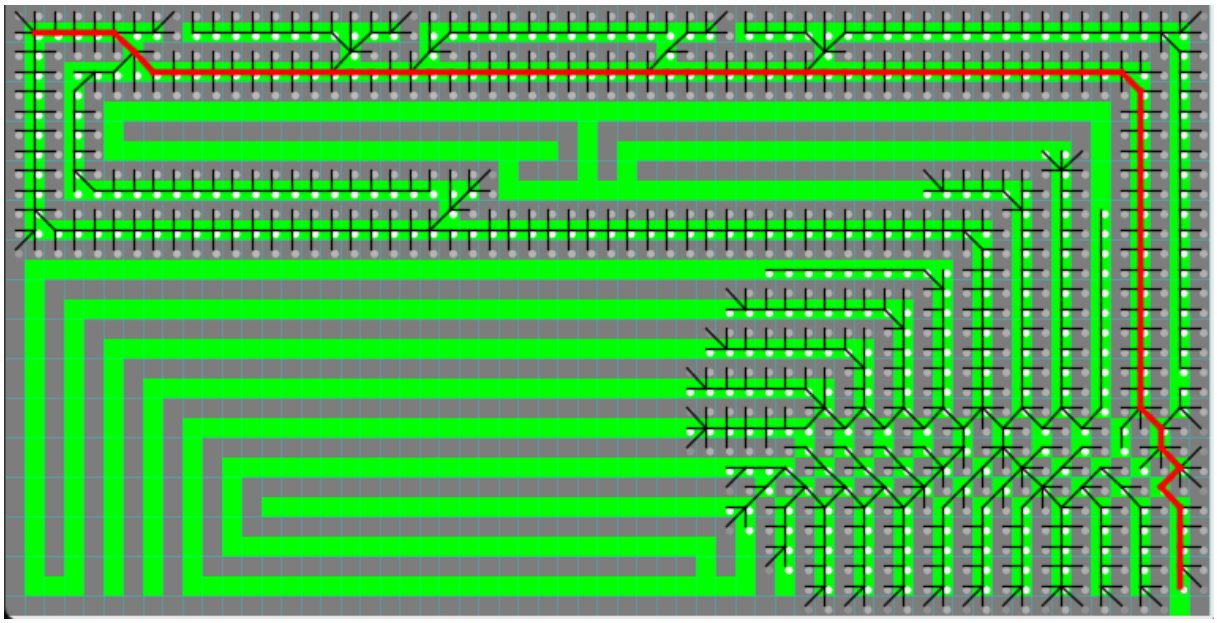

Fig. 9. – Exécution de Dijkstra sur le labyrinthe

4.4.2 Exécution de A*

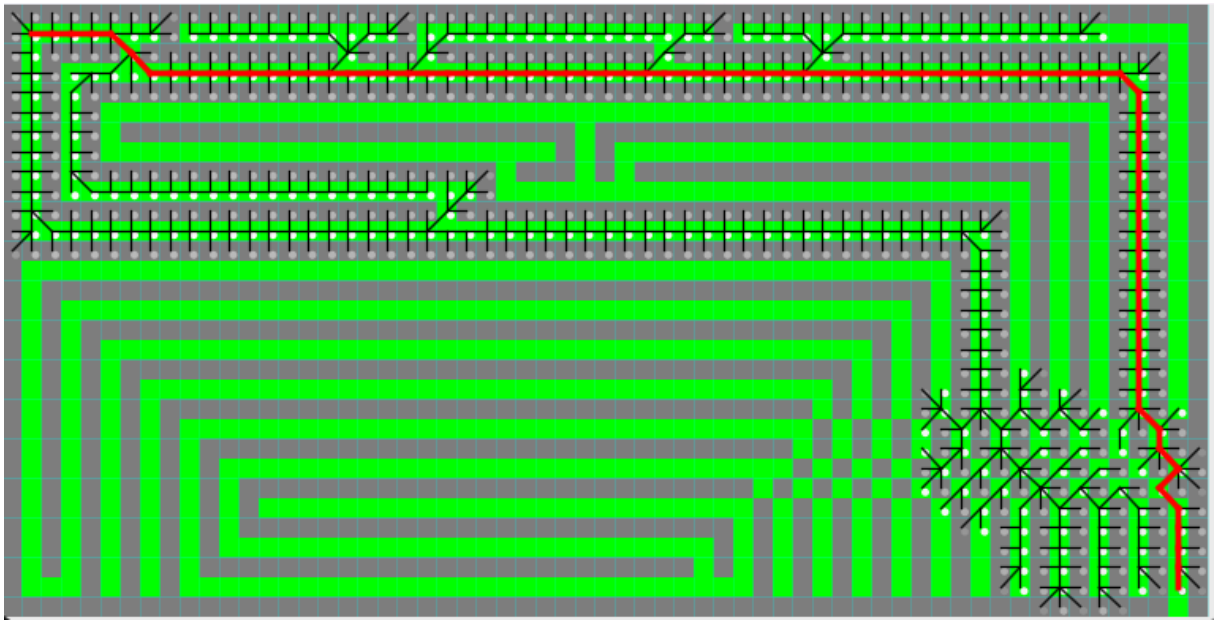

Fig. 10. – Exécution de A* (euclidien) sur le labyrinthe

4.4.3 Comparaison

Algorithme	Nœuds explorés	Coût du chemin	Optimal ?
Dijkstra	437	83.89949493661166	Oui
A* (Euclidien)	275	83.89949493661166	Oui
A* (Manhattan)	86	83.89949493661166	Non
A* (Chebyshev)	285	83.89949493661166	Oui

Tableau 4. – Comparaison des algorithmes sur le labyrinthe

Analyse : On observe aussi une réduction significative du nombre de nœuds explorés avec A*.

4.5 Synthèse des résultats

Carte	Dijkstra	A* Eucl.	A* Manh.	A* Cheb.
Standard	4156	4130	4107	4135
Uniforme	5000	2844	100	3484
Labyrinthe	437	275	86	285

Tableau 5. – Nombre de nœuds explorés par algorithme et par carte

Conclusions principales :

1. **Dijkstra** explore systématiquement le plus de nœuds car il n'a aucune information sur la destination
2. **A* avec heuristique admissible** (Euclidien, Chebyshev) garantit l'optimalité tout en réduisant l'exploration
3. **A* avec Manhattan** est le plus rapide mais peut trouver des chemins sous-optimaux en 8-connexité
4. Le **gain de A*** dépend fortement de la topologie

5 Discussion et Cas d'Utilisation Réels

5.1 Analyse des performances

5.1.1 Dijkstra

Avantages :
- Garantit toujours le chemin optimal
- Simple à implémenter
- Ne nécessite pas de connaissance sur la destination

Inconvénients :
- Explore uniformément dans toutes les directions
- Inefficace quand la destination est connue

5.1.2 A*

Avantages :
- Exploration dirigée vers l'objectif
- Généralement plus rapide que Dijkstra
- Optimal si l'heuristique est admissible

Inconvénients :
- Nécessite une bonne heuristique
- Performance dépendante de la qualité de l'heuristique

5.2 Cas d'utilisation réels

5.2.1 Navigation GPS

Les systèmes GPS utilisent des variantes de A* avec des heuristiques basées sur :
- Distance à vol d'oiseau
- Estimation du temps basée sur les vitesses moyennes

A* est préféré car la destination est toujours connue.

5.2.2 Routage réseau

Pour le routage dans les réseaux informatiques :
- Dijkstra est souvent utilisé
- La destination n'est pas toujours connue à l'avance
- Calcul des tables de routage vers toutes les destinations

5.2.3 Robotique

En robotique mobile :

- A* pour la planification de trajectoire
- Heuristiques tenant compte des contraintes physiques
- Souvent combiné avec des algorithmes de planification locale

6 Conclusion

6.1 Bilan

Ce projet nous a permis d'implémenter et de comparer deux algorithmes fondamentaux de recherche du plus court chemin. Les résultats expérimentaux confirment les propriétés théoriques :

- **Dijkstra** explore de manière uniforme et garantit l'optimalité
- **A*** avec une heuristique admissible réduit significativement le nombre de nœuds explorés tout en conservant l'optimalité
- Le choix de l'heuristique influence fortement les performances de A*

6.2 Difficultés rencontrées

- Gestion correcte des poids diagonaux
- Choix d'heuristiques admissibles
- La création de cartes prend beaucoup de temps.

6.3 Améliorations possibles

- Utilisation d'une file de priorité (tas binaire) pour réduire la complexité
- Support de cartes plus complexes (terrain 3D, images)

7 Annexes

7.1 Instructions d'utilisation

7.1.1 Compilation et exécution

```
# Compilation
javac -encoding UTF-8 -d bin src/up/MainApp/*.java

# execution
java -cp bin up.MainApp.App
```

7.1.2 Exécution avec jar

```
# Aide
java -jar PathFinder.jar --help

# Dijkstra (defaut)
java -jar PathFinder.jar graphe.txt

# A* avec heuristique euclidienne
java -jar PathFinder.jar -a astar graphe.txt

# A* avec heuristique de Manhattan
java -jar PathFinder.jar -a astar -h manhattan graphe.txt

# A* avec heuristique de Chebyshev
java -jar PathFinder.jar -a astar -h chebyshev graphe.txt
```