Rapport mini Projet

BASCI Onur
numéro d'étudiant : 22002054

18 Décembre 2022

Algorithmique Avancée

Université Paris Cité
UFR de Mathématiques-Informatique
2022-2023

Contents

1 Introduction

Pour le projet de cours d'Algorithmique Avancée, j'ai développé un logiciel qui répond à l'exigence demandée par le scénario 2 "Graphes et Image (Dijkstra et modélisation)". Il s'agit ici de représenter une image comme un graphe orienté et valué. Les sommets de ce graphe représentent les pixels et les arêtes représentent la différence d'intensité. Le but de projet est en utilisant l'algorithme de Dijakstra, trouver et dessiner le plus court chemin entre ces deux pixels. Il faut aussi si possible d'optimiser l'algorithme afin d'obtenir un résultat plus efficace. L'algorithme Dijakstra calcule tous les chemins plus courts en partant d'un sommet pour aller à tous les nœuds. En conséquence cela peut résulter des longs durée d'execution. Pour cela, le logiciel propose une deuxième méthode inspiré de l'algorithme A étoile. Cette fonction ne trouve pas exactement le chemin plus court, mais il fait une bonne estimation de plus court chemin à l'aide d'une fonction d'heuristique. Le logiciel nous permet de change l'intensité des coefficients entre deux pixels. Avec cela, nous pouvons vois l'effet des poids pour ces algorithmes.

2 Le logiciel

2.1 utilisation

Quand on ouvre l'application on rencontre avec une page d'acceuil d'intérface.

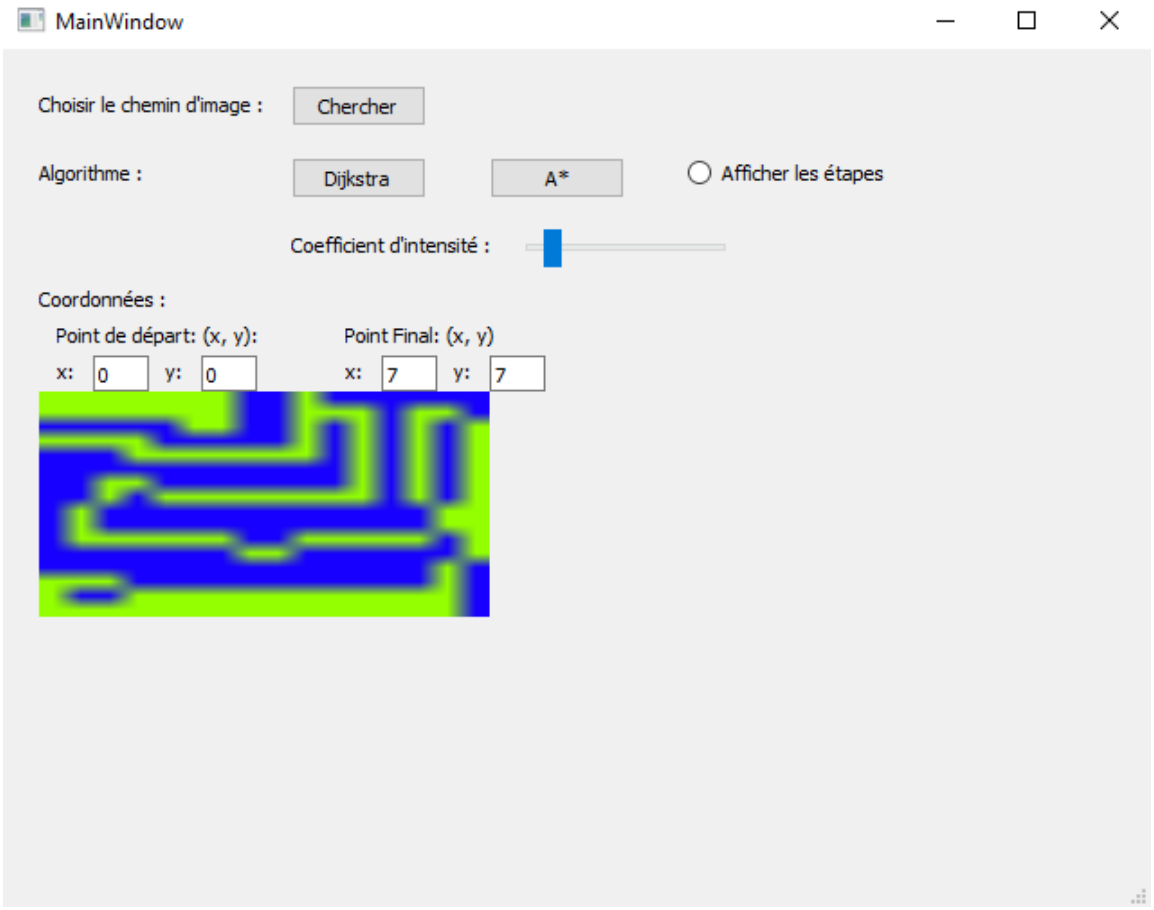

Figure 1: Page d'acceuil du logiciel

Ici le logiciel nous propose plusieurs fonctionnalités :

- Le bouton " chercher " vous permet de parcourir dans votre explorateur Windows. Vous pouvez choisir une image de votre préférence pour appliquer les algorithmes.
- Le bouton radio " Afficher les étapes " vous permet de montrer toutes les étapes lors de l'application de l'algorithme. Vous pouvez le fermer si vous voulez voir le résultat directement.
- Le bar " Coefficient d'intensité " change les poids des arêtes. Vous pouvez le changer pour voir l'effet des poids pour les algorithmes.
- "Les coordonnées" vous permet de saisir le point (pixel) de départ et de fin. X représente les lignes, Y représente les colonnes.
- Le Buton "Dijkstra" vous permet d'appliquer l'algorithme de Dijkstra sur l'image choisit (l'image par défaut sinon). Le chemin obtenu sera affiche dans une fenêtre séparée.
- Le Buton "A*" vous permet d'appliquer l'algorithme de A étoile sur l'image choisit (l'image par défaut sinon). Le chemin obtenu sera affiché dans une fenêtre séparée.

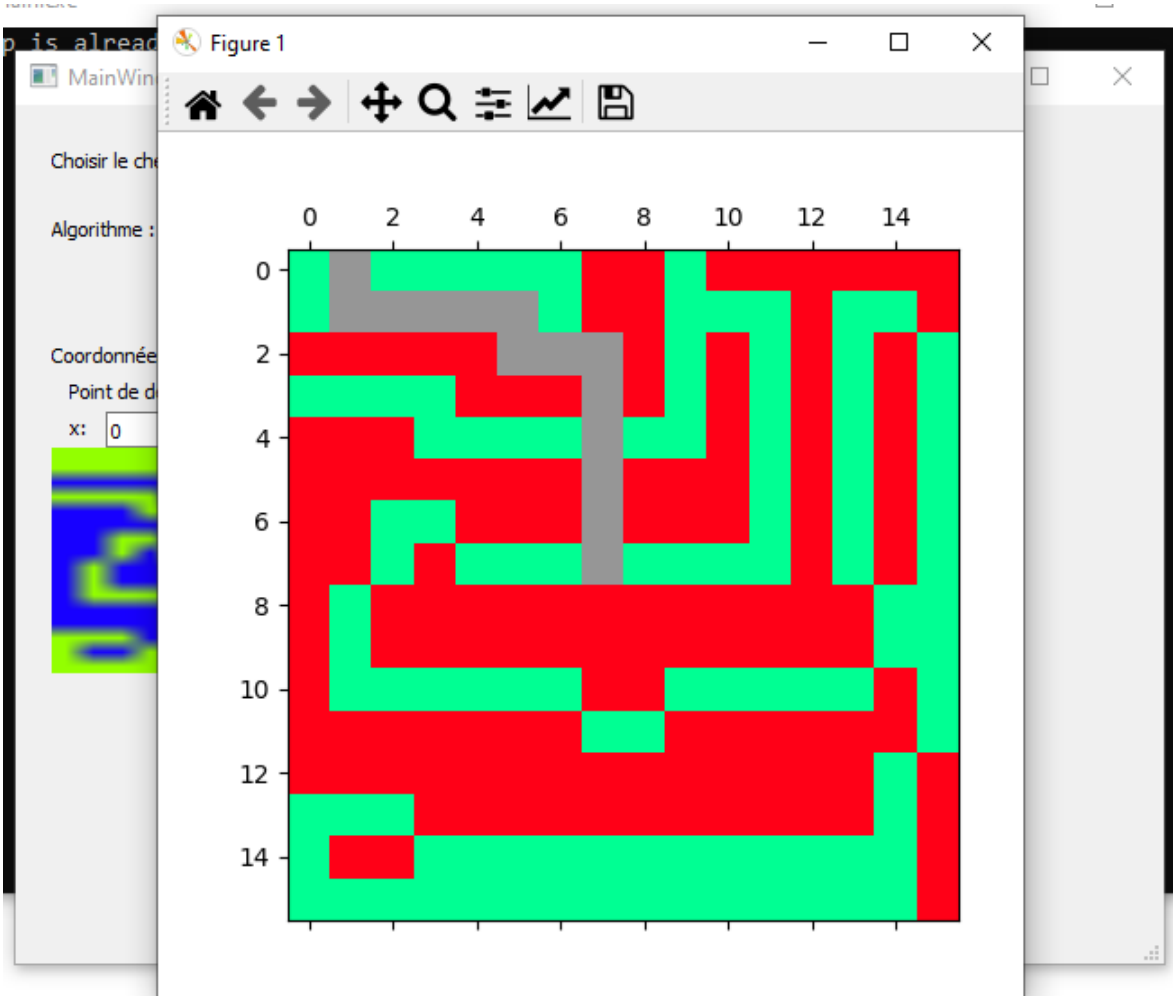

Figure 2: Fenêtre montrant le chemin obtenu en fonction de A*

3 Implémentation

Le logiciel nous propose 2 Algorithme principale , Dijkstra et A étoile

3.1 Les structures de données

3.1.1 Graphe

Dans ce projet, je représente un graphe avec une liste d'adjacences. Pour cela, j'utilise la classe graphe qui contient un attribut m_adj_list qui est un dictionnaire.

3.1.2 Arbre

Dans ce projet, j'utilise aussi la structure arbre que j'utilise pour obtenir l'arbre couvrant à partir de Dijakstra. Je représente un arbre avec la classe Tree, qui possède un attribut children contenant les fils du nœud.

3.1.3 Représentation d'un image comme un graphe

Dans ce projet je représente une image comme un graphe, avec les noeuds sont des pixels et les arêtes sont la différence d'intensité. Pour les arêtes j'utilise la 4 connexité et la distance entre

deux pixel est défini en fonction de l'image. J'utilise une simple différence pour les images en noir et blanc avec

$$|Couleur1 - Couleur2|$$

. Vu que, dans ce cas, les pixels sont entre 0-255 cela ne pose pas de problème. Et pour les images colorés j'utilise la distance euclidienne

$$\sqrt{(color1.R - color2.R)^2 + (color1.G - color2.G)^2 + (color1.B - color2.B)^2}$$

À cause de la 4 connexité nous avons donc de $nombre_de_pixel * 4$ arêtes dans le graphe.

4 Implémentation

Le logiciel nous propose 2 Algorithme principales , Dijkstra et A étoile.

4.1 Dijkstra

Pour implémenter l'algorithme de Dijakstra je me suis inspirée de cours d'Algorithmique Avancée. J'utilise le pseudo-code proposé dans le cours.

Data: Un graphe connexe valué G de fonction de poids ω et un sommet u
Result: Un arbre couvrant T et la distance $D(v) = d_G(u, v)$ pour tout v

$T = \{u\}$
for $v \in V(G)$ **do**
  $dist\_prov(v) := \infty$; $pere(v) := \emptyset$
end
$dist\_finale(u) := 0$; $dernier\_ajout := u$

while $V(T) \neq V(G)$ **do**
  for v *voisin de* $dernier\_ajout$ **do**
    if $dist\_finale(dernier\_ajout) + \omega(dernier\_ajout, v) < dist\_prov(v)$ **then**
      $dist\_prov(v) :=$
        $dist\_finale(dernier\_ajout) + \omega(dernier\_ajout, v)$
      $pere(v) := dernier\_ajout$
    end
  end
  Selectionner $v \notin V(T)$ tel que $dist\_prov$ est minimum
  Ajouter $(pere(v), v)$ a T
  $dist\_finale(v) := dist\_prov(v)$
  $dernier\_ajout := v$
end

Algorithm 6: Algorithme de Dijkstra

Figure 3: Dijkstra proposé en cours

4.2 A étoile

Après avoir implémenté l'algorithme Dijkstra j'ai remarqué qu'elle prend du temps considérable. Surtout pour les images à partir de 64*64, cela devient même Inexécutable. On va voir plus précisément cela dans la partie complexité, mais j'ai eu l'impression que cette durée d'exécution

vient du fait que l'algorithme Dijkstra calcule la distance pour chaque pixel séparément. Pour résoudre ce problème, j'ai décidé d'implémenter un deuxième algorithme qui utilise 2 fonctions, heuristique et poids, pour décider quel pixel à parcourir. Cet algorithme est inspiré par le cours. CS 50 Introduction to Artificiel Intelligence. L'avantage de cet algorithme est ce qu'il ne parcourt pas tous les nœuds, mais il choisit le nœud à parcourir à partir d'une fonction d'heuristique et du poids. Ici, j'utilise la distance Manhattan qui est calculée avec la différence d'indices du pixel courant et du pixel cherché.

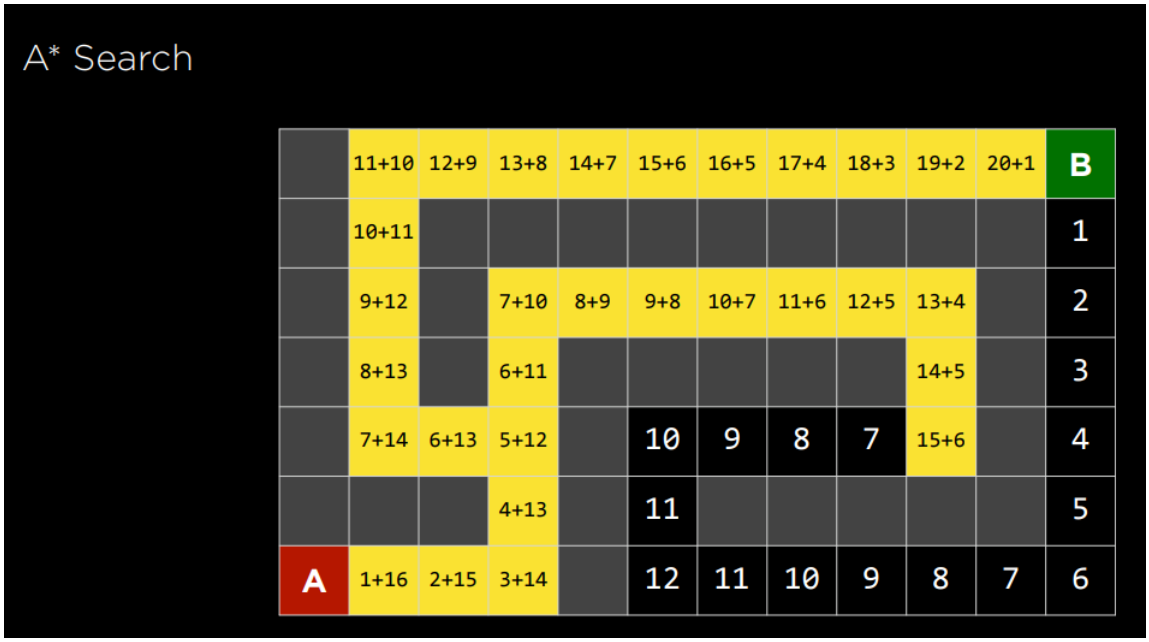

Figure 4: Exemple de l'application de l'algorithme pris de CS 50 Introduction to Artificiel Intelligence

Comme vous pouvez voir dans l'exemple les cases commençant de 1 à 7 ne sont pas parcourus.

5 Complexité

5.1 Dijkstra

Lors de l'implémentation de l'algorithme de Dijkstra, je n'ai pas fait une différence par rapport au pseudo-code proposé en cours donc la complexité de cet algorithme est $O(n^2)$

5.2 Dijkstra

Dans le pire de cas, cet algorithme parcourt tous les pixels donc c'est de complexité $O(n)$ avec n nombre de pixels. Et sinon la complexité $< O(n)$.

5.3 Conclusion

5.4 Durée d'éxecution

À partir de quelques essais sur les différentes images, nous remarquons que l'algorithme A* s'exécute sans un problème même pour les images de 1000×1000 pixels. Alors que l'algorithme de Dijkstra ne fonctionne que jusqu'aux images de 64×64 pixels sur mon ordinateurs personel.

Nous pouvons en conclure que l'objectif demandé est obtenu avec l'algorithme A* au niveau de temps d'exécution.

5.5 Efficacité

5.6 Exemple 1

Par sa définition, l'algorithme Dijkstra nous donne sûrement le chemin plus court alors que l'algorithme A* nous donne une estimation. Voilà quelque comparaison obtenue avec le logiciel pour comparer l'efficacité.

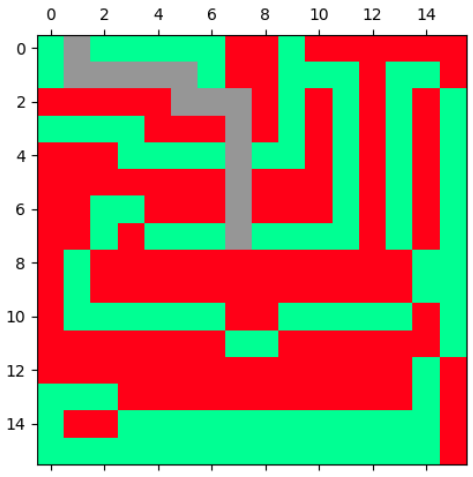

Figure 5: Le chemin obtenu à partir de A*

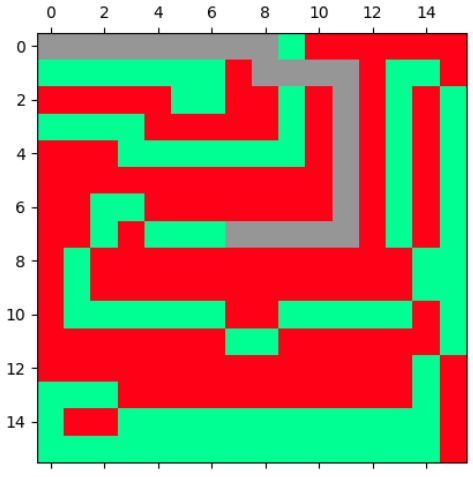

Figure 6: Le chemin obtenu à partir de dijkstra

Comme nous pouvons voir l'algorithme A*, échappe d'abord les cases rouges (qui ont plus de poids) jusqu'à la case de colonne 7 qui est identique à la case finale. Cela pourquoi il change de la direction et il va directement vers la case final. Pour l'algorithme de Dijkstra, on voit qu'au lieu de déscendre directement de la case 7, (il fait une sorte de semi-cercle pour accéder.) la case finale. Ici, le chemin choisit par l'algorithme Dijkstra est meilleur puisqu'il

fait 2 changements de couleur (colonne 6 à 7 et 8 à 9) alors que l'algorithme A* fait 4 (Ligne 2 à 3,3 à 4, 4 à 5 et 6 à 7). Ici, le passage de cases de la même couleur n'a pas de coût (voir la définition de la distance).

5.7 Exemple 2

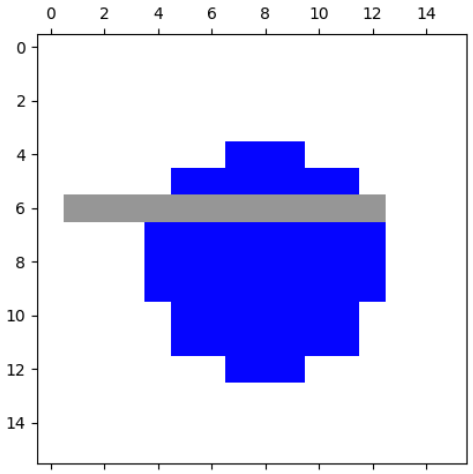

Figure 7: Le chemin obtenu à partir de A* avec intensité = 1

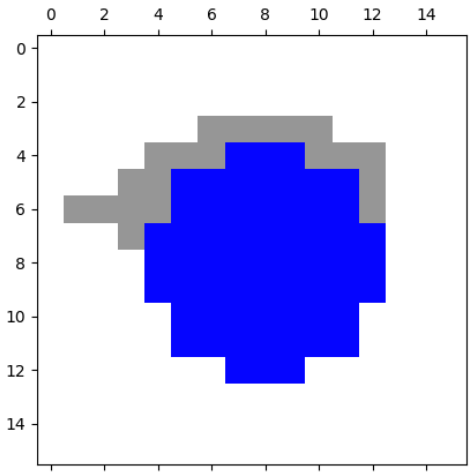

Figure 8: Le chemin obtenu à partir de A* avec intensité = 8

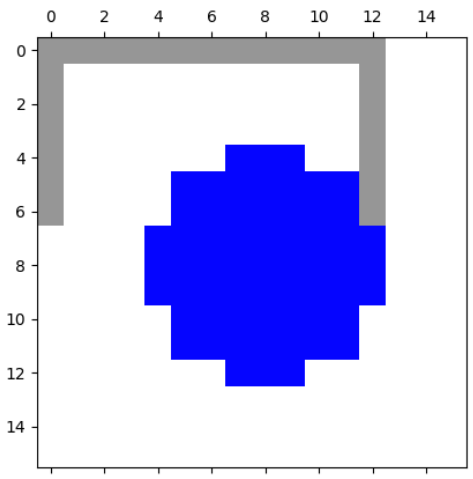

Figure 9: Le chemin obtenu à partir de Dijkstra

Ici, nous remarquons que l'algorithme A* marche mieux avec le coefficient d'intensité plus élevé. En fait, quand l'intensité est plus élevée, l'algorithme préfère plutôt de ne pas changer de la couleur. Dans la figure 8, vous pouvez voir les pixels découverts par l'algorithme. Nous voyons qu'au lieu de passer par le pixel bleu, il a choisi de descendre même si le pixel en bas est heuristiquement plus éloigné. Nous remarquons dans la figure 9 que le chemin passe par le haut, mais il faut se rappeler que comme ces cases sont de la même couler le poids de ce chemin est 0.

5.8 Bilan

À partir des exemples, nous avons vu que l'algorithme A* diminue considérablement la durée d'exécution. Et au niveau d'efficacité, il est plus efficace lorsqu'on augmente l'intensité de différences entre les pixels.