Projet : Fast Marching

Redouane BOUAZZA , Thomas Boeno , Bahaeddine HILAL

16/01/2022

Contents

1 Introduction

La méthode du fast marching est utilisée pour suivre l'évolution d'un front d'onde et peut être utilisée pour résoudre des problèmes de mécanique des fluides ou de détection de contour par exemple. Le principe n'est pas d'étudier la vitesse du front d'onde mais d'établir les temps de passage de ce front à des points prédéfinis de l'espace que l'on appelle maillage. Cette méthode permet de résoudre l'équation d'Eikonal, de la fome $|\nabla \mathrm{T}| = \mathrm{F}$.

Après l'execution de l'algorithme nous obtenons une matrice de temps de passage du front en chaque point du maillage. L'application pour trouver le plus court chemin au sein d'une image est directement issue de cette matrice en utilisant une méthode de descente du gradient à partir du point d'arrivée souhaité.

2 Partie Théorique

L'objectif de la méthode Fast Marching est de calculer numériquement le temps d'arrivée T, solution de l'équation stationnaire:

$$c(x)\,|\nabla_x T(x)| = 1$$

On se place en dimension 2 d'espace $x = (x_1, x_2)$ sur un maillage de pas d'espace $h = (\Delta x_1, \Delta x_2)$, et on propose la discrétisation suivante de l'équation stationnaire

$$max(\frac{T_{i,j} - T_{i-1,j}}{\Delta x_1}, -\frac{T_{i+1,j} - T_{i,j}}{\Delta x_1}, 0)^2 + max(\frac{T_{i,j} - T_{i,j-1}}{\Delta x_2}, -\frac{T_{i,j+1} - T_{i,j}}{\Delta x_2}, 0)^2 = \frac{1}{c^2(x_{i,j})}$$

Pour mettre à jour la frontière à chaque itération, et en posant $\begin{cases} t_1 = T_{i-1,j} \\ t_2 = T_{i+1,j} \\ t_3 = T_{i,j-1} \\ t_4 = T_{i,j+1} \end{cases}$

on est amené à résoudre pour θ l'équation:

$$max(\frac{\theta - t_1}{\Delta x_1}, -\frac{t_2 - \theta}{\Delta x_1}, 0)^2 + max(\frac{\theta - t_3}{\Delta x_2}, -\frac{t_4 - \theta}{\Delta x_2}, 0)^2 = \frac{1}{c^2(x_{i,j})}$$

En posant $h_1 = \Delta x_1$, $h_2 = \Delta x_2$, $v_1 = min(t_1, t_2)$ et $v_2 = min(t_3, t_4)$, l'équation s'écrit alors:

$$\frac{1}{h_1^2}(\theta - min(v_1, \theta))^2 + \frac{1}{h_2^2}(\theta - min(v_2, \theta))^2 = \frac{1}{c^2}$$

Ainsi dans le cas $h = h_1 = h_2$:

- Si $max(v_1, v_2) - min(v_1, v_2) < \frac{h}{c}$

$$\theta = \frac{v1 + v2}{2} + \frac{1}{2}\sqrt{-(v_2 - v_1)^2 + \frac{2h^2}{c^2}}$$

- Si $max(v_1, v_2) - min(v_1, v_2) \geq \frac{h}{c}$

$$\theta = min(v_1, v_2) + \frac{h}{c}$$

3 Applications:

3.1 Application à la recherche du plus court chemin dans un labyrinthe:

Dans un premier temps, nous avons testé notre programme sur un labyrinthe pour trouver le plus court chemin pour aller d'un point initial choisi à un autre. Le labyrinthe est initialement sous une forme d'image (.png,.jpeg...), nous avons donc converti l'image en une matrice où chaque case représente un pixel qui ne peut prendre que deux valeurs: 0 pour le noir et 1 pour le blanc.

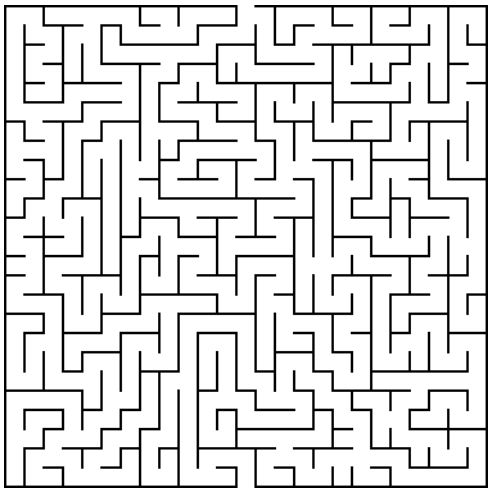

Figure 1: Image du labyrinthe choisi pour ce test

Cependant, afin de pénaliser le passage à travers les bords du labyrinthe nous avons établi la matrice de contrainte suivante, où on affecte la valeur 0.001 aux pixels blancs et 1 aux pixels noirs.

En appliquant ensuite l'algorithme du fast marching pour cette métrique en choisissant l'entrée du labyrinthe située en haut milieu comme point initial, nous obtenons la fonction T: distance au sommet initial par rapport à notre métrique, representé par le graphe ci dessous :

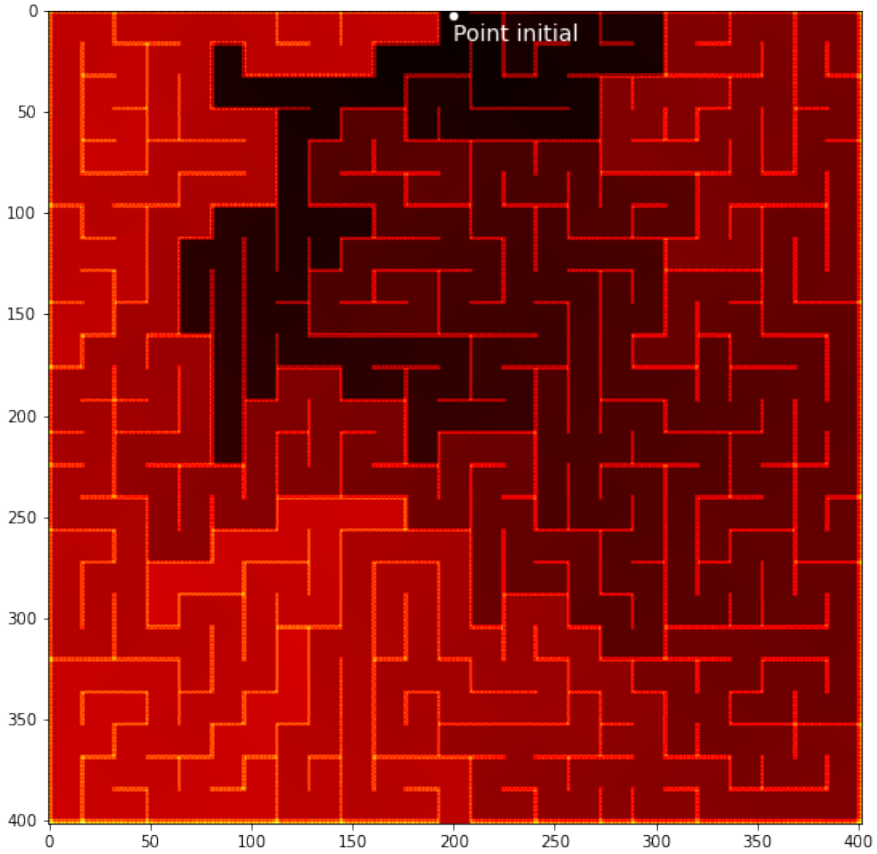

Figure 2: Distance au point inital marqué par le point blanc

En utilisant la matrice T trouvée précedemment, le chemin le plus entre le point de départ et le point de final peut être trouvé. Ceci est faisable en utlisant la méthode de descente du gradient en partant du point final, le chemin correspond donc aux ensemble de points obtenus au cours de chaque itération de la descente du gradient. On remarque que le chemin ne passe pas à travers les bords du labyrinthe, ceci est dû au choix de métrique que nous avons pris initialement.

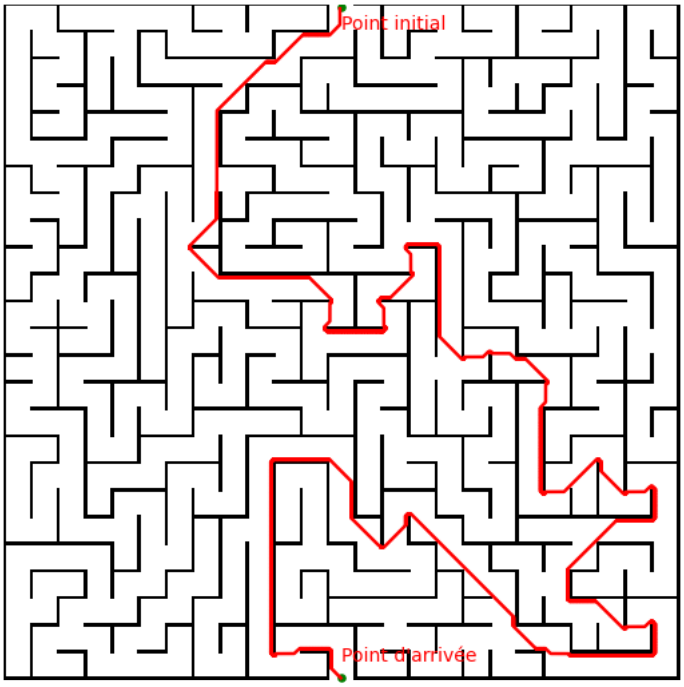

Figure 3: Application descente de gradient

3.2 Application planificateur d'itinéraire:

Comme deuxième appliaction du Fast Marching, nous avons réalisé un planificateur d'itinéraire qui détermine le chemin le plus court d'un point de la France à un autre. Pour cela nous avons importé une carte basique du résau routier francais, les lignes bleu ciel correspondent aux autoroutes où la vitesse moyenne est de 120km/h et les lignes rouges correspondent aux routes nationales où la vitesse moyenne est 80km/h et le reste qui est blanc correspond au reste du territoire national.

Figure 4: Traitement de la carte du réseau

Par le même principe que le labyrinthe, nous avons établi une matrice de contrainte pour restreindre prioritairement le déplacement aux autoroutes et routes nationales. Pour cela nous avons tout d'abord converti cette image couleur en une image en niveau de gris, puis nous avons affecté la valeur 1/120 aux pixels qui tendent vers le gris (précedemment bleu ciel), 1/80 aux pixels qui tendent vers le noir (précédemment rouge) et 1 au reste qui est blanc.

En appliquant ensuite l'algorithme du fast marching pour cette métrique et en choisissant la ville de Brest comme point initial, nous obtenons la fonction T: distance au sommet initial par rapport à notre métrique, représenté par le graphe ci dessous:

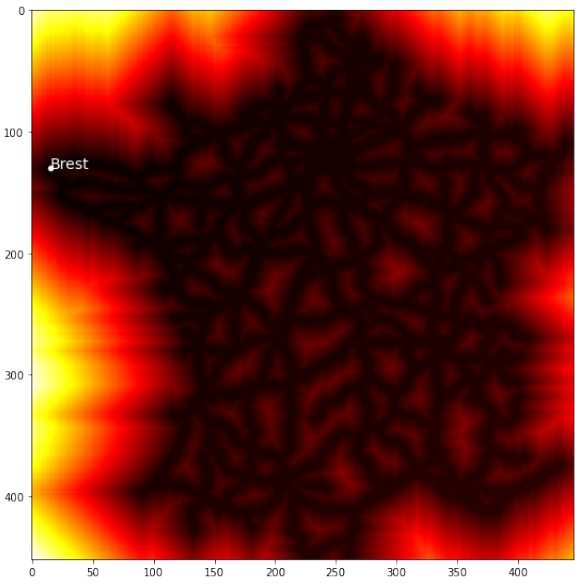

Figure 5: Distance au point inital marqué par le point blanc

En utilisant la matrice T trouvée précedemment et la méthode du descente de gradient, nous pouvons trouver le trajet le plus court entre “Brest” et “Besançon”.

En comparant le résultat obtenu avec celui obtenu avec GoogleMaps, on remarque que les deux routes ont plus ou moins la même allure avec quelques nuances ce qui est logique puisque le réseau routier utilisé par google est bien plus complet et riche que celui que nous avons utilisé pour notre programme.

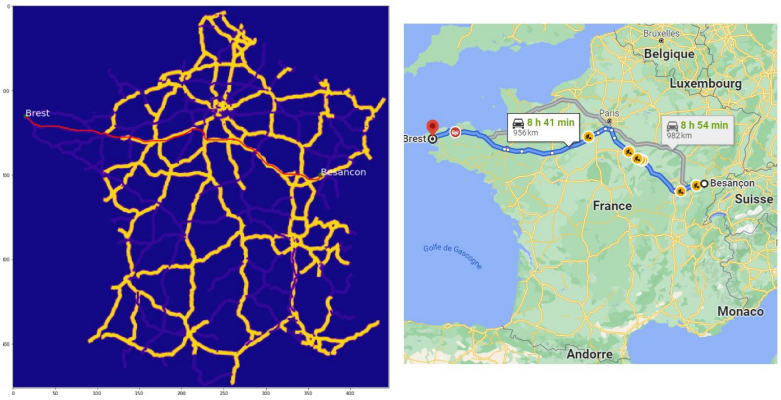

Figure 6: Le plus court chemin entre Brest et Besançon

4 Conclusion

En concluant, les résultats sur le labyrinthe sont concluant. Cependant, nous pouvons critiquer le fait que le chemin trouvé a tendance à longer les murs ce qui semble peu réaliste. Afin de résoudre ce problème, nous pouvons ajouter un malus aux abords des obstacles.

En ce qui concerne l'application sur les trajets routiers, ici aussi les résultats sont proches de la référence que constitue Google Maps, tout en sachant que l'unique donnée prise en compte est la limitation de vitesse sur les routes et non le trafic routier et les travaux.

5 Références bibliographiques

1. Explication Fast Marching Université de Berkley
2. Implementing Fast Marching Algorithm
3. Slides cours fast marching