Algorithmes de recherche d'itinéraires

Samuel ARSAC

4918

1 Présentation du problème

Nous disposons d'un graphe du réseau routier d'une région extrait de la base de données OpenStreetMap. Les nœuds de ce graphe représentent les intersections et les arêtes les rues ou routes. Nous disposons des coordonnées de chaque nœud, ainsi que de la longueur, du nom de la voie et de la limite de vitesse pour chaque arête. Les fichiers extraits sont au format graphml, et nous les avons transformés en une base SQL contenant les informations importantes. Plutôt que de pondérer le graphe par les longueurs des rues, nous avons divisé ces longueurs par la limite de vitesse sur la voie, dans le but de rendre les itinéraires plus adaptés à une optimisation du temps de parcours.

L'objectif est donc d'obtenir un algorithme efficace prenant en argument un tel graphe, un nœud de départ et un nœud d'arrivée et déterminant un itinéraire optimal (c'est-à-dire de poids total minimal) entre ces deux nœuds.

Dans le but tester ces algorithmes, nous utiliserons quatre itinéraires :
- A : court itinéraire dans le graphe de Lyon
- B : itinéraire traversant le graphe de Lyon
- C : itinéraire dans le graphe du département du Rhône n'empruntant pas d'autoroute
- D : itinéraire traversant le graphe du Rhône, utilisant principalement l'autoroute

2 Algorithme de Dijkstra

L'algorithme de Dijkstra est un algorithme classique de parcours de graphes, explorant à chaque itération le nœud non exploré le plus proche du nœud depuis lequel la recherche est effectuée. On conserve en mémoire le nœud précédant chaque nœud dans le parcours, et on s'arrête lorsque le nœud d'arrivée doit être examiné. Il est facile de voir que l'itinéraire alors trouvé, en parcourant en sens inverse la mémoire des précédents depuis le nœud d'arrivée, est optimal : s'il existait un itinéraire plus court, le nœud d'arrivée aurait été examiné à une itération antérieure.

L'inconvénient de cet algorithme est la dispersion de la recherche dans toutes les directions, ce qui peut mener à explorer un très grand nombre de nœuds. La complexité de cet algorithme est un $O(A + N \log N)$ (avec A le nombre d'arêtes et N le nombre de nœuds du sous-graphe examiné), la file de priorité permettant de choisir le nœud suivant possédant une structure de tas-min permettant une modification du poids d'un élément en temps logarithmique. Cela correspond

dans le cas d'un graphe de degré borné à un $O(N \log N)$. Ainsi, plus le nombre de nœuds examinés est grand, plus le temps de calcul sera important. Les résultats (figure 1) montrent un temps de calcul de 3,5 secondes pour l'itinéraire D, ce qui est trop important pour être utilisé dans un logiciel, surtout pour un itinéraire de cette taille.

3 Algorithme A*

L'algorithme A* utilise le même principe que l'algorithme de Dijkstra, mais a recours à une heuristique pour estimer la proximité d'un nœud avec le nœud d'arrivée. Le nœud exploré à chaque itération est le nœud connu et non visité ayant la somme de sa distance au départ et de sa distance estimée à l'arrivée la plus faible. La recherche est donc dirigée vers le nœud d'arrivée.

Se pose la question de l'heuristique à utiliser. La première idée est d'utiliser la distance euclidienne, mais une autre possibilité est une heuristique basée sur des «balises» (*landmarks*). Il y a toutefois une contrainte : l'heuristique ne doit pas surestimer la distance réelle, sans quoi l'optimalité de l'itinéraire trouvé n'est plus garantie.

3.1 Distance euclidienne

Connaissant les coordonnées des nœuds, il est aisé de calculer la distance euclidienne. Toutefois, le poids des arêtes est un temps, il faut donc diviser par une vitesse. Or l'heuristique doit sous-estimer le poids, on divise donc la distance calculée par la limite de vitesse maximale, 130 km/h.

On observe que les temps de calcul pour les différents itinéraires sont plus élevés qu'avec l'algorithme de Dijkstra (figure 1). Le nombre de nœuds explorés a pourtant bien diminué grâce à l'utilisation d'une heuristique, mais la mesure du temps de calcul pour un seul nœud montre que le calcul de l'heuristique fait doubler ce temps pour A* par rapport à Dijkstra. La diminution du nombre de nœuds n'est alors pas suffisante pour compenser cette augmentation du temps de calcul.

3.2 Distance euclidienne prenant en compte les autoroutes

La faible diminution du nombre de nœuds est sans doute imputable à la grande sous-estimation des poids par l'heuristique précédente (distance divisée par 130 km/h). On peut donc imaginer une heuristique comparant deux estimations de poids : la première sans passer par les autoroutes, en divisant cette fois la distance euclidienne par 90 km/h, et une deuxième estimation consistant en la somme de la distance à l'entrée d'autoroute la plus proche du départ, la distance euclidienne divisée par 130 km/h jusqu'à la sortie la plus proche de l'arrivée, et la distance de cette sortie à l'arrivée (figure 2). L'heuristique prend le minimum de ces deux estimations (la plus grande des deux pouvant être plus grande que le poids réel).

Pour ce faire on utilise l'algorithme de Dijkstra (cette fois pour parcourir le graphe) en partant d'un nœud imaginaire à distance 0 des entrées d'autoroutes et en parcourant le graphe dans le sens inverse (le trajet devant provenir du nœud et non pas de l'entrée), ce qui permet de calculer du même coup l'entrée

la plus proche de chaque nœud et la distance du nœud à celle-ci. On fait ensuite de même pour les sorties. L'entrée la plus proche n'est toutefois pas nécessairement l'entrée menant à la bonne direction sur l'autoroute, mais il n'y a pas de problème d'optimalité puisque dans tous les cas l'heuristique sur l'autoroute minore le temps de trajet (division par 130 km/h).

Les résultats sont toutefois encore plus décevants que pour la distance euclidienne simple (figure 1). Encore une fois, il semble que l'augmentation du temps de calcul imposée par l'heuristique ne soit pas compensée par la réduction du nombre de nœuds explorés.

3.3 Balises

La dernière heuristique utilise le principe suivant : on utilise un ensemble de nœuds, des «balises», desquels on connaît la distance à chaque autre nœud dans le graphe non orienté associé au graphe de départ. Pour chaque paire de nœuds (A, B) et balise D, on peut donc calculer $|d(D, B) - d(D, A)|$ (les distances étant calculées dans le graphe non orienté). Or cette valeur est nécessairement inférieure à $d(A, B)$: si on suppose $d(D, B) > d(D, A)$, supposons que l'on ait $d(A, B) < d(D, B) - d(D, A)$, alors $d(D, A) + d(A, B) < d(D, B)$, ce qui est absurde, puisque $d(D, B)$ est la distance sur un itinéraire optimal. Si on prend le maximum de ces valeurs sur toutes les balises, on obtient donc la minoration la plus proche de $d(A, B)$ avec cet ensemble de balises.

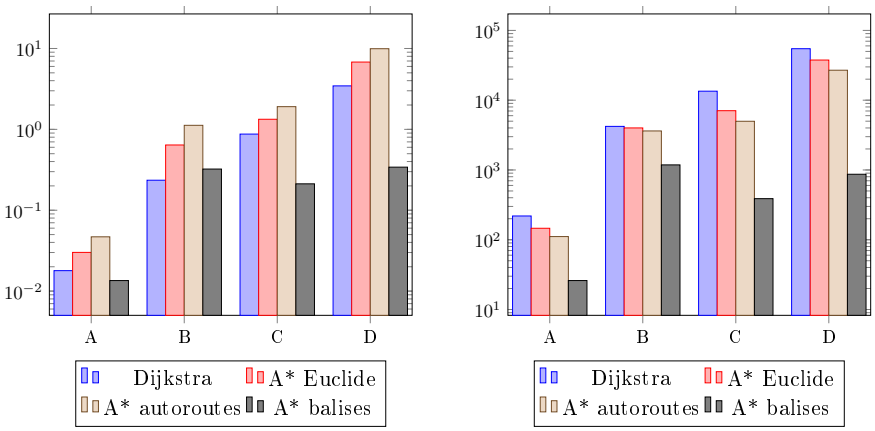
Pour déterminer ces balises, nous utilisons pour chacune une heuristique basée sur l'arbre de parcours du graphe par l'algorithme de Dijkstra à partir de l'ensemble des précédentes (pour la première, on initialise avec un nœud choisi sans autre règle particulière que d'être sur la principale composante connexe du graphe). Cette heuristique affecte à chaque nœud de l'arbre un poids qui correspond à la distance du nœud au départ à laquelle on ajoute les poids de ses fils. On parcourt l'arbre en se dirigeant toujours vers le fils de poids maximal, et la feuille ainsi sélectionnée devient la balise suivante. On obtient de cette façon les 10 balises indiquées sur la figure 3.

On observe que les temps de calcul ont tendance à se stabiliser quand le nombre de balises augmente (figure 4). On peut supposer que lorsqu'une balise est particulièrement bien placée par rapport à l'itinéraire, le temps de calcul est significativement réduit, mais il diminuera peu ensuite car les nouvelles balises n'auront que peu de chances d'être mieux placées. Les résultats de cette heuristique sont concluants pour les itinéraires A, C et D, toutefois le temps de calcul pour B reste supérieur à celui par l'algorithme de Dijkstra. On observe que le temps de calcul par nœud reste élevé, toutefois la diminution du nombre de nœuds (figures 5 et 6) explorés est suffisante pour rendre le temps de calcul plus faible qu'avec l'algorithme de Dijkstra (figure 1). Si on calcule le rapport du temps de calcul par l'algorithme de Dijkstra et du meilleur temps de calcul par A* avec les balises, on obtient : A : 1,33, B : 0,73, C : 4,12, D : 10,11.

4 Conclusion

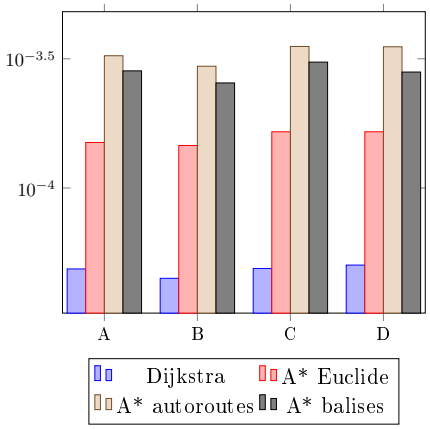
Les résultats montrent que l'algorithme de Dijkstra est limité dans le cadre de la recherche d'itinéraires, le temps de calcul augmentant trop rapidement avec la longueur de l'itinéraire à calculer. Les performances de l'algorithme A*

dépendent de l'heuristique choisie. On observe que les heuristiques basées sur la distance euclidienne ne sont pas suffisament efficaces, toutefois l'heuristique basée sur les balises est en général plus efficace que l'algotihme de Dijkstra, particulièrement sur les longs trajets.



(a) Temps de calcul (s)

(b) Nombre de nœuds explorés



(c) Temps de calcul par nœud (s)

FIGURE 1 – Comparaison entre tous les algorithmes (les valeurs pour A* avec les balises sont celles pour le meilleur nombre de balises)

FIGURE 2 – Illustration de l'heuristique séparant les autoroutes

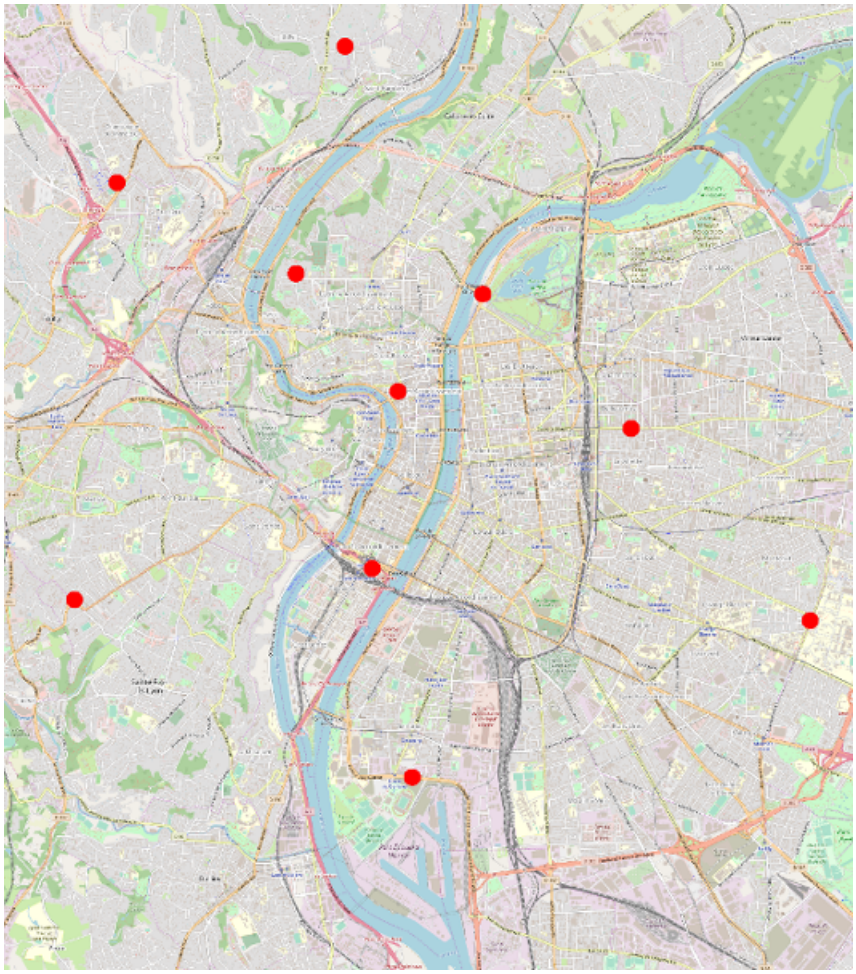
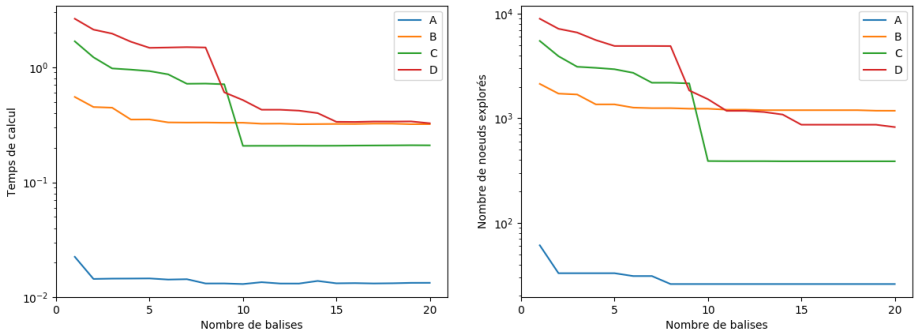


FIGURE 3 – Positionnement de 10 balises sur le grpahe de Lyon



(a) Temps de calcul (s)

(b) Nombre de nœuds explorés

FIGURE 4 – Résultats de A* avec les balises, en fonction de leur nombre

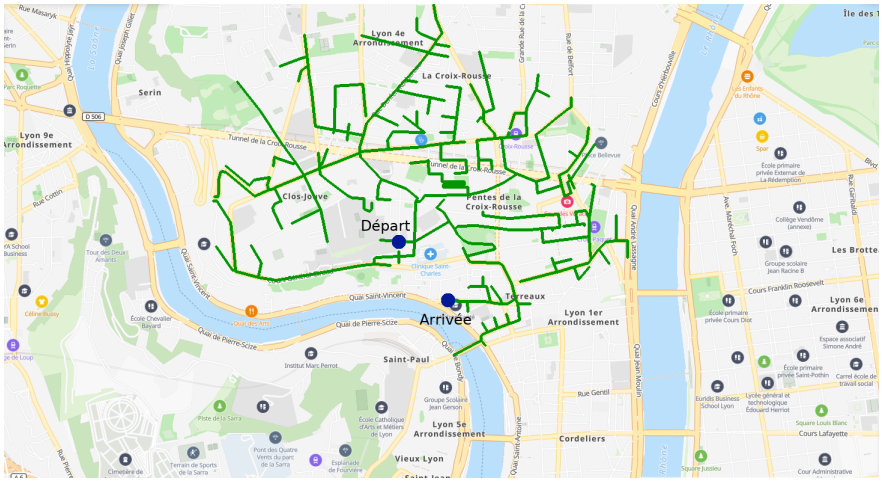


FIGURE 5 – Noeuds découverts par l'algorithme de Dijkstra pour l'itinéraire A

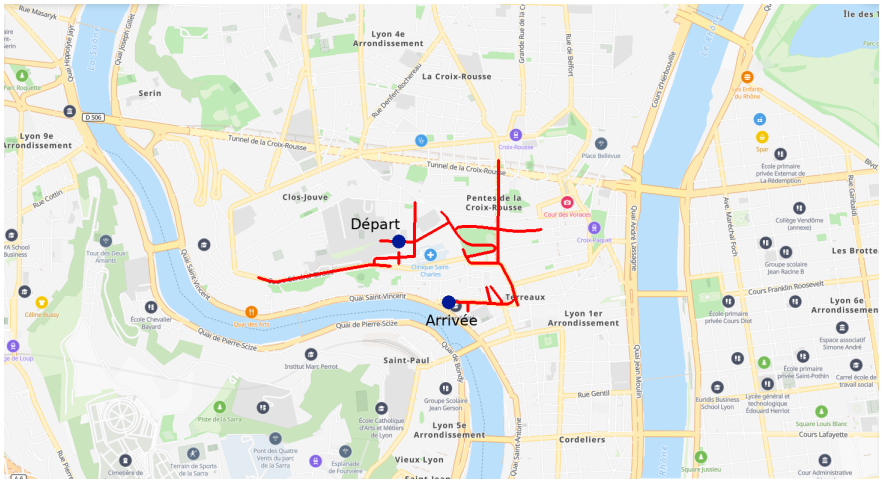


FIGURE 6 – Noeuds découverts par l'algorithme A* avec 20 balises pour l'itinéraire A

Algorithme		Itinéraire	Temps de calcul	Nœuds explorés	Temps de calcul par nœud
Dijkstra		A	0,0179 s	219	$4,86 * 10^{-5}$ s
		B	0,235 s	4213	$4,47 * 10^{-5}$ s
		C	0,874 s	13459	$4,88 * 10^{-5}$ s
		D	3,449 s	54681	$5,03 * 10^{-5}$ s
A*	Euclidienne	A	0,0301 s	146	$1,50 * 10^{-4}$ s
		B	0,641 s	4003	$1,46 * 10^{-4}$ s
		C	1,333 s	7071	$1,65 * 10^{-4}$ s
		D	6,795 s	37616	$1,65 * 10^{-4}$ s
	Autoroutes	A	0,0469 s	111	$3,25 * 10^{-4}$ s
		B	1,122 s	3624	$2,96 * 10^{-4}$ s
		C	1,909 s	4987	$3,53 * 10^{-4}$ s
		D	9,938 s	26885	$3,52 * 10^{-4}$ s
	Balises	A	0,0135 s	26	$2,84 * 10^{-4}$ s
		B	0,323 s	1182	$2,55 * 10^{-4}$ s
		C	0,212 s	388	$3,07 * 10^{-4}$ s
		D	0,341 s	868	$2,81 * 10^{-4}$ s

FIGURE 7 – Résultats numériques correspondants à la figure 1