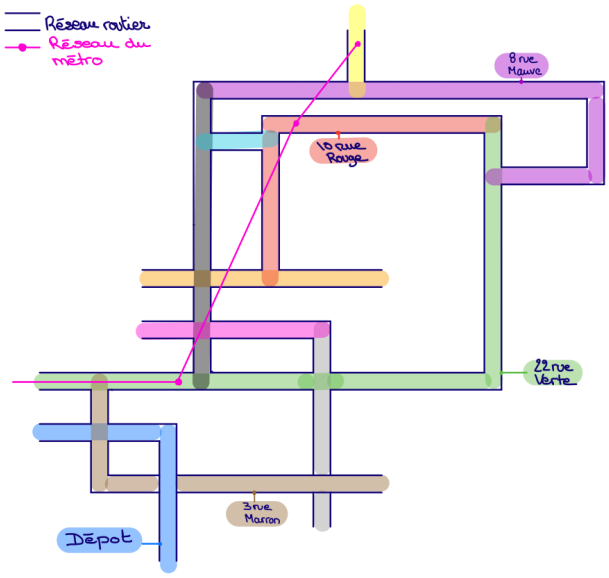
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

Java

Projet de Recherche Opérationnelle

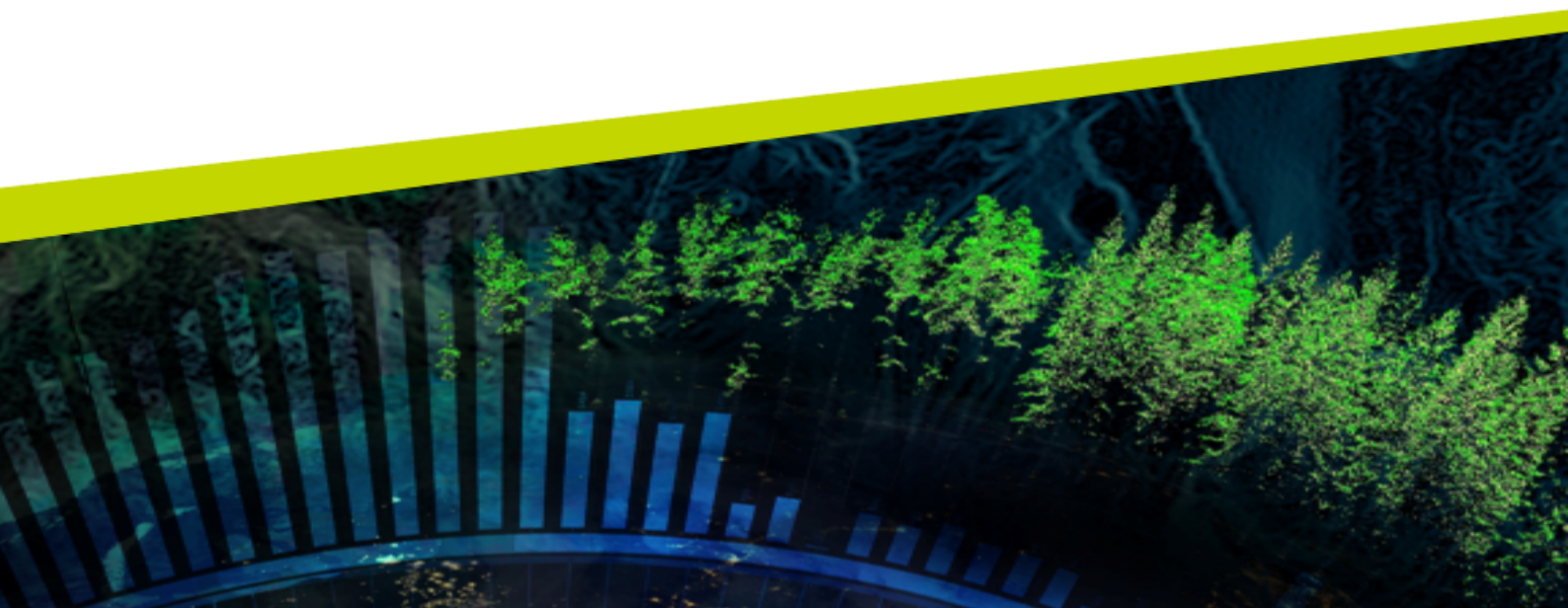
- Balade en ville

Etien Niels
De Crescenzo Ambre
ITI4



Délivré le 8 Novembre, 2022

Professeur référent : Geraldine Del Mondo

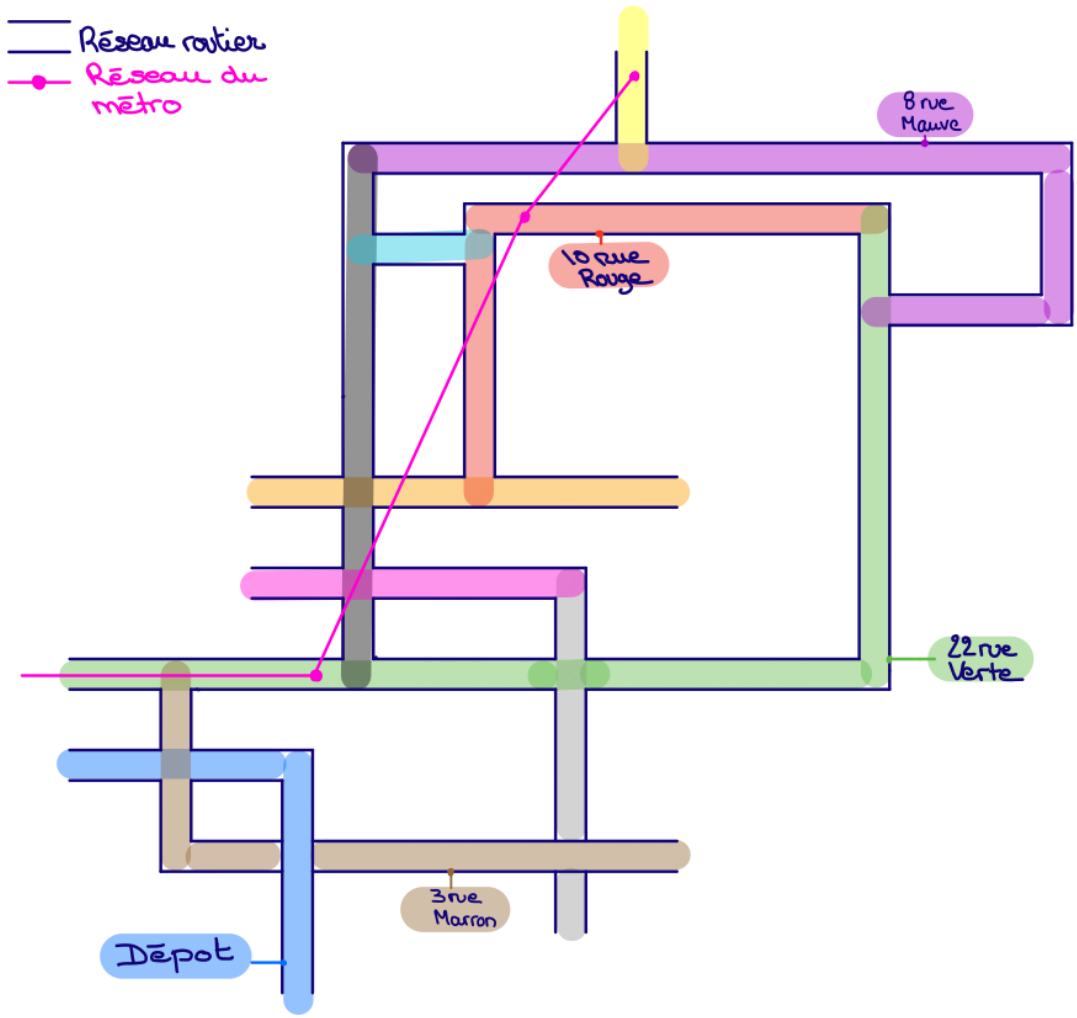


Sommaire

1 Introduction

Le but de ce projet de Recherche opérationnelle est de répondre à un problème de livraison. Le dépot est situé au 5 de la rue Bleu sur le plan de la ville ci-dessous. Elle dispose également d'un réseau de métro utilisable pour les livraisons. On souhaite obtenir le trajet le plus court et qui ne repasse pas deux fois par la même adresse. Le trajet de livraison standard passe via quatre adresses de la ville, respectivement au 10 rue Rouge, 8 rue Mauve, au 22 rue Verte et 3 rue Marron avant de revenir au dépôt.

Notre objectif est donc d'implémenter une fonction affichant le meilleur trajet (en terme de distance), ne passant pas 2 fois par la même adresse ; il faut partir du dépôt, passer une fois par chaque adresse, avant de finalement revenir au dépot.



2 Modélisation, définition du problème formel, et association d'une classe de complexité

2.1 Modélisation

2.1.1 Graphe originel

$G = (V, E, l_e)$
$V = \{s_i | i \in [|intersections| + |adresses|]\}$
$E = \{(c_j, c_k)$ | ∃ une rue/métro entre ces intersections/adresses ; $j \neq k$ et $j, k \in [|V|]$; et $c_j, c_k \in V\}$
$l_e : E \to R^+$
$l_e(c_j, c_k) = longueur(c_j, c_k)$

Caractéristiques :

- non orienté
- arcs valués
- connexe
- simple

2.1.2 Graphe intermédiaire (pour résoudre le problème)

$G = (V, E, l_e, ch_e)$
$V = \{s_i | i \in [|adresses|]\}$
$E = \{(c_j, c_k)$ | ∃ un chemin entre ces adresses ; $j \neq k$ et $j, k \in [|V|]$; et $c_j, c_k \in V\}$
$l_e : E \to R^+$
$l_e(c_j, c_k) = longueur(c_j, c_k)$
$ch_e : E \to R^+$
$ch_e(c_j, c_k) = chemin(c_j, c_k)$

Caractéristiques :

- non orienté
- arcs valués
- connexe
- simple

2.2 Problème formel

Problème :
Trouver un cycle de coût minimal passant par toutes les adresses de livraison + le dépôt.

Transformation du problème pour qu'il soit résolvable : il faut transformer le graphe contentant les intersections et les adresses en un graphe contenant seulement les adresses.

Le nouveau problème devient le suivant :
Trouver le chemin hamiltonien de coût minimal dans ce nouveau graphe, s'il existe.

2.3 Complexité

La classe de complexité associée à ce genre de problème est non polynomiale, NP-Complet. On pense que cela devrait bien se passer puisque dans notre exemple, nous n'avons que 5 adresses (le dépot + 4 adresses de livraison), on peut donc vraisemblablement trouver la solution dans un temps raisonnable.

3 L'algorithme

3.1 Etat de l'art

3.1.1 Algorithme du plus court chemin

Dijkstra : Algorithme polynomial. Il prend en entrée un graphe dont les arcs sont valués (avec des valeurs positives) et un sommet, et retourne un sous-graphe dans lequel les sommets sont valués avec leur distance au sommet de départ. Le but final de cet algorithme est de récupérer la distance minimale entre le sommet de départ, et les autres sommets du graphe.
Bellman-Ford : Cet algorithme permet de calculer les plus courts chemins depuis un sommet source, les chemins pouvant avoir des valeurs négatives. Il ne nous intéresse pas dans notre projet puisque toutes les valuations des arcs sont des distances, qui sont positives.
A* : Il permet de rechercher le chemin le plus court entre deux sommets d'un graphe, en prenant en compte d'éventuels obstacles. Il utilise une évaluation heuristique afin de trouver le meilleur chemin. Dans notre cas, on préfèrera utiliser Dijkstra puisque l'on a pas besoin d'éviter des obstacles.

3.1.2 Voyageur de commerce

Algorithme glouton : La plus simple à mettre en place : Une heuristique gloutonne permet d'obtenir une solution, trouvée par une suite de décisions définitives sans retour arrière. Il existe par exemple la méthode du plus proche voisin, celle de la plus proche insertion, la plus lointaine insertion et la meilleure insertion.
Algorithme de Christofides : C'est un algorithme d'approximation du problème du voyageur de commerce. Il est ne donne pas la solution optimale, mais est très efficace. Pour la taille de notre graphe, on préfèrera donc utiliser le premier algorithme au lieu de celui-ci.

3.1.3 Bibliographie

fr.wikipedia.org

3.2 Pseudo-code

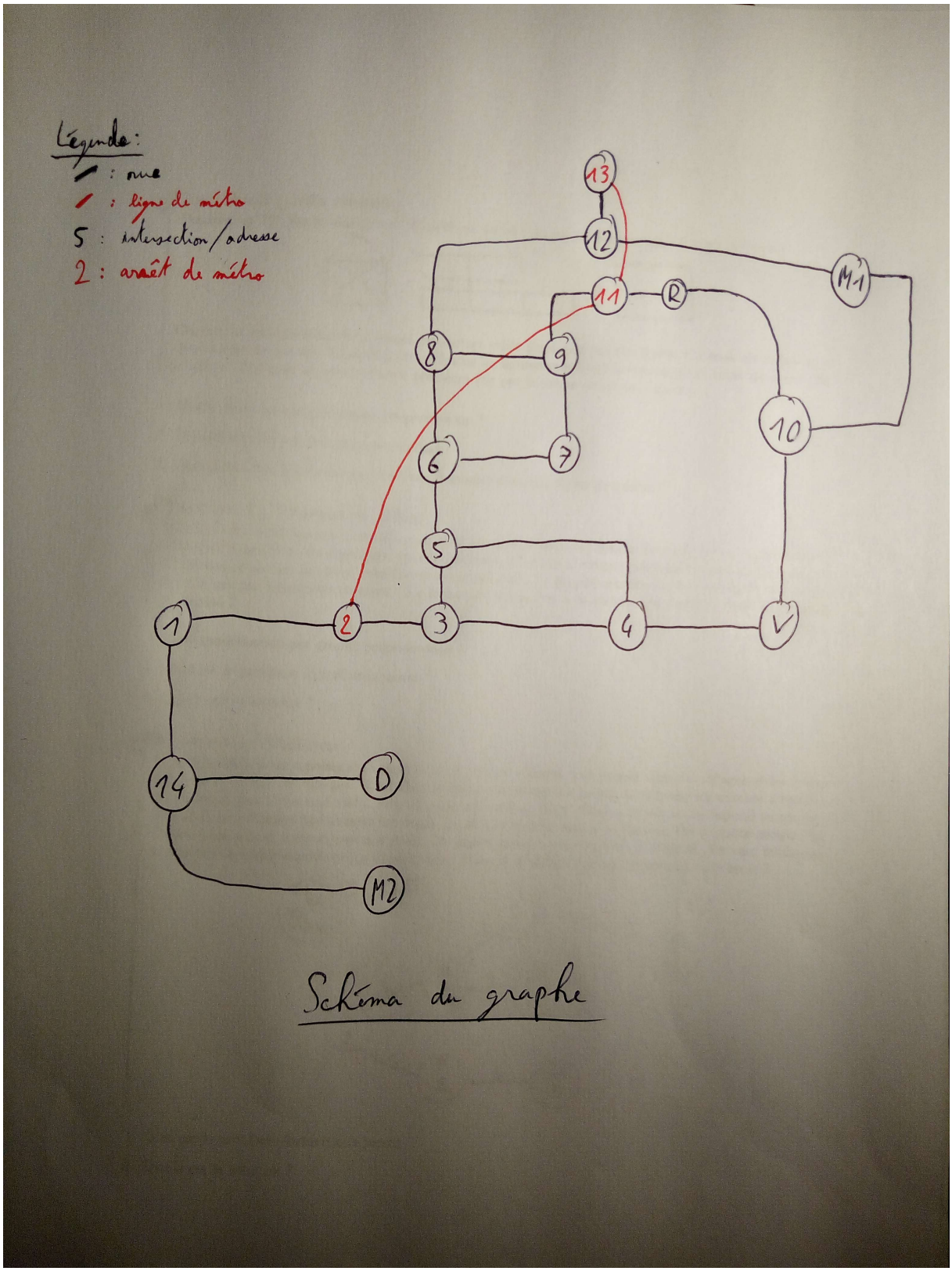
3.2.1 Algorithme de Dijkstra

```
Procédure Initialisation(G,sdeb)
    pour chaque point s de G faire
        d[s] := infini /* on initialise les sommets autres que sdeb à infini */
    fin pour
    d[sdeb] := 0 /* la distance au sommet de départ sdeb est nulle */
Fin
Procédure Trouve_min(Q)
    mini := infini
    Sommet := -1
    pour chaque sommet s de Q
        si d[s] < mini
        alors
            mini := d[s]
            sommet := s
    renvoyer sommet
Fin
Procédure maj_distances(s1,s2)
    si d[s2] > d[s1] + Poids(s1,s2) /* Si la distance de sdeb à s2 est plus grande que */
    alors
        d[s2] := d[s1] + Poids(s1,s2) /* On prend ce nouveau chemin qui est plus court */
        prédécesseur[s2] := s1 /* En notant par où on passe */
Fin
Procédure Dijkstra(G,sdeb)
    Initialisation(G,sdeb)
    Q := ensemble de tous les nœuds
    tant que Q n'est pas un ensemble vide faire
        s1 := Trouve_min(Q)
        Q := Q privé de s1
        pour chaque nœud s2 voisin de s1 faire
            maj_distances(s1,s2)
        fin pour
    fin tant que
Fin
```

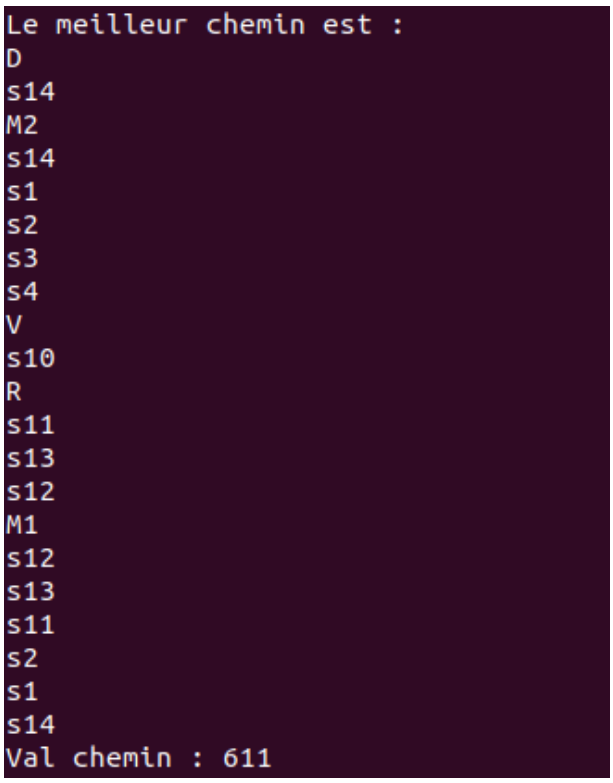
Algorithme 1: Algorithme de Dijkstra

4 Implémentation

4.1 Graphe des données



4.2 Résultat



Le programme nous affiche une suite de sommets, qui correspond à la route optimale (en terme de distance) à emprunter afin passer par toutes les adresses (V,R,M1,M2) en partant du dépot (D), ainsi que la distance à parcourir.

5 Adaptation

5.1 Adaptation : métro fermé

Première idée : ajouter un attribut booléen EstMétro sur les sommets, que l'on initialise lors du chargement des données. Si 2 sommets qui possèdent EstMétro à VRAI sont emprunté consécutivement dans le résultat final de notre algorithme, alors le trajet emprunte le métro, sinon non.

Seconde idée : créer un fichier métros.txt qui contient les noms des arrêts de métro. Si 2 sommets ayant des noms présents dans métros.txt sont emprunté consécutivement dans le résultat final de notre algorithme, alors le trajet emprunte le métro, sinon non.

Troisième idée : créer un fichier semblable à celui du réseau, mais sans le métro. Et on demande ensuite à l'utilisateur si Oui ou Non (O/N) il souhaite que le trajet emprunte le métro. On peut aussi simplement créer un second main correspondant au cas où le métro est fermé (qui charge le second graphe ; sans métro donc).

5.2 Panne du métro de la ville

S'il y a une panne sur le métro, il nous suffit d'appliquer la 3ème idée pour se débarasser du problème : charger un fichier ne contenant pas les sommets correspondants aux arrêts de métro.
Dans notre programme, nous avons choisi de faire deux classes exécutables : BaladeEnVilleAvecMetro et BaladeEnVilleSansMetro.

```
Procédure main()
    ...
    GrapheListePlusCourtChemin g ← GrapheListePlusCourtChemin.deFichier("reseau.txt");
    ...
Fin
Procédure main()
    ...
    GrapheListePlusCourtChemin g ← GrapheListePlusCourtChemin.deFichier("reseauSansMetro.txt");
    ...
Fin
```

Algorithme 3: Début de l'algorithme du main de BaladeEnVilleSansMetro

```
Le meilleur chemin est :
D
s14
M2
s14
s1
s3
s4
V
s10
R
s10
M1
s12
s8
s6
s5
s3
s1
s14
Val chemin : 647
```

Résultat du programme sans métro.

6 Conclusion

Ce projet nous a permis de mettre en pratique les connaissances amassées au cours du semestre, notamment les méthodes de résolution des problèmes de graphes : définir le graphe, poser le problème, faire un état de l'art, proposer une solution du problème, puis l'implémenter. Nous n'avons pas eu de problème majeur durant ce projet, il n'était néanmoins pas facile à réaliser. La difficulté principale du projet résidait dans l'implémentation du code, la compréhension du code déjà écrit dans les précédents travaux pratiques, ainsi que dans l'organisation et la répartiton du travail.