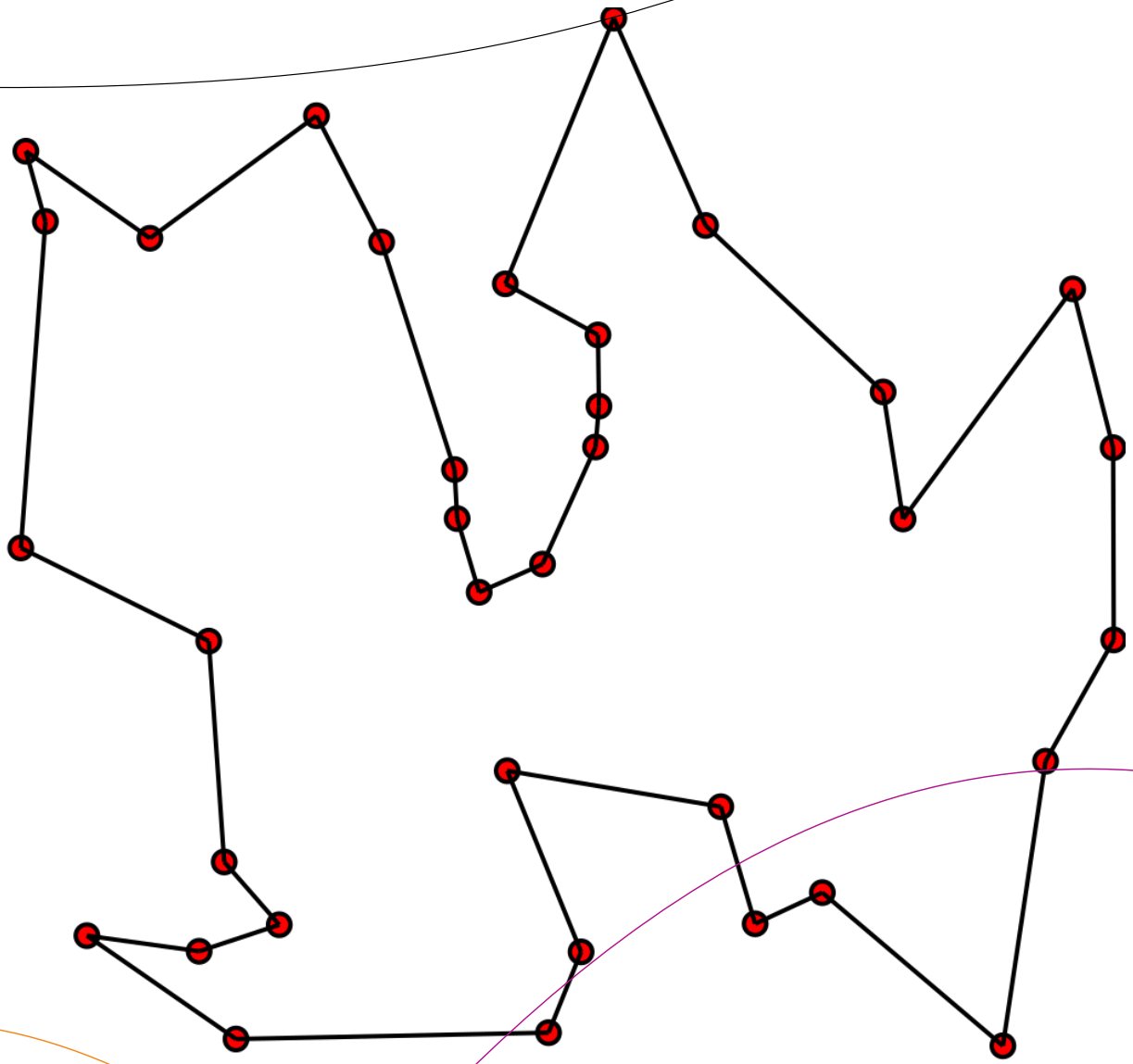
INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
ROUEN NORMANDIE



Mathias Hersent
Ryan Juteau

Projet RO

Le voyageur de commerce

RO - ITI4 - 2022

Table des matières

1 Introduction

1.1 Présentation du projet

L'objectif de ce projet est de développer un algorithme de résolution d'un problème simple du voyageur de commerce avec tous les outils que nous avons vus en cours et des recherches bibliographiques adéquates.

2 Etape 1 : Modéliser, définir le problème formel, associer une classe de complexité

2.1 Question 1 : Modélisation et caractéristiques

Les problèmes formels soulevés seront développés dans la deuxième question.
Voici les propriétés du premier graphe que nous avons trouvé en étudiant le problème :

- Non orienté
- Non complet
- Trois types de noeuds différents mais les intersections de rues ou de métro ont le même comportement.

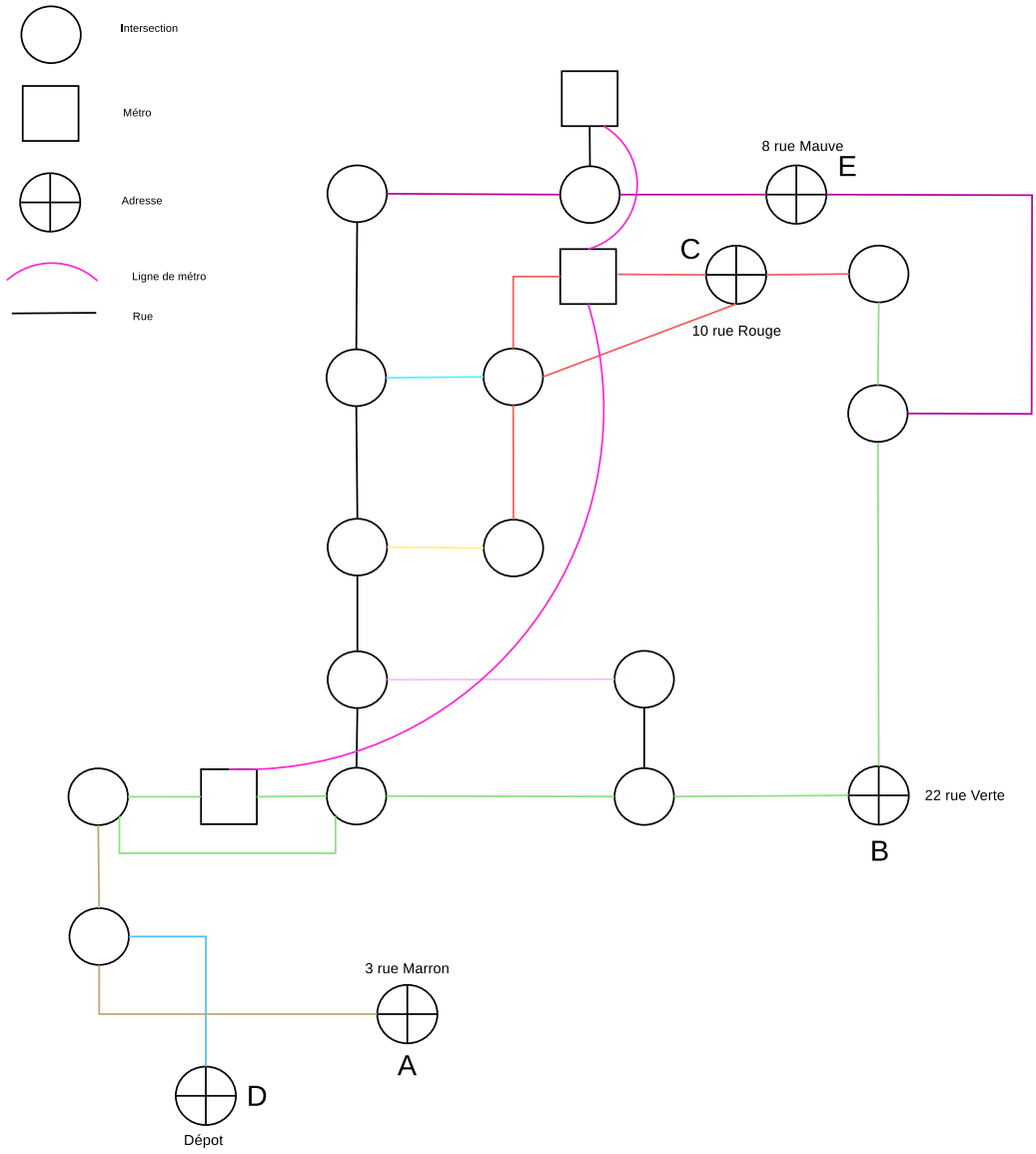


FIGURE 1 – Représentation sous forme de graphe du problème

2 ETAPE 1 : MODÉLISER, DÉFINIR LE PROBLÈME FORMEL, ASSOCIER UNE CLASSE DE COMPLEXITÉ

Le deuxième graphe a été créé à partir du premier, en voici les propriétés :
- Non orienté
- Complet
- Un noeud symbolise une adresse
- Une arête est le chemin le plus court entre deux adresses

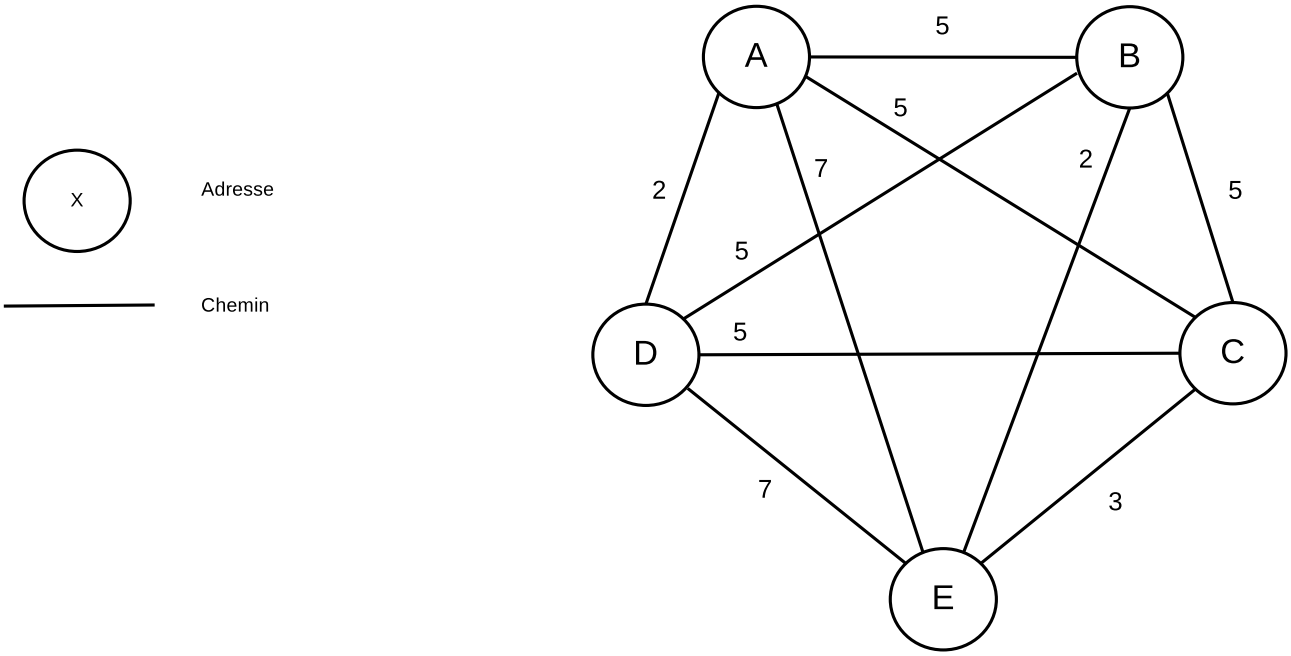


FIGURE 2 – Représentation sous forme de graphe du deuxième problème

2.2 Question 2 : Problèmes formels

Le problème est celui du voyageur de commerce. En effet, il faut trouver un cycle hamiltonien qui passe par les adresses, mais il doit avoir un coût minimal.

Actuellement, il faut prévoir deux types de noeud : intersection de rues et d'adresses, mais nous avons rajouté la distinction entre rue et métro à l'avance. On peut repasser plusieurs fois par une intersection de rue ou de métro, mais pas d'une adresse. De plus, si le métro est indisponible, il faut pouvoir le retirer facilement du graphe et n'utiliser que les intersections de rues. Il a été aussi décidé de matérialiser les adresses sous forme de noeuds, sinon impossible d'aller 3 rue marron et d'y revenir si cela avait été une arête. Il aurait fallu en plus matérialiser les fins de rues. Cela permet d'aller à une adresse et de rebrousser chemin si besoin. Les arêtes permettent de compter le coût de déplacement puisqu'emprunter une arête coûte 1.

Un deuxième graphe a été créé à partir du premier. Il s'agit d'un graphe complet avec le coût pour aller d'une adresse à une autre, où chaque noeud est une adresse et chaque arête est le coût du plus court chemin.

2.3 Question 3 : Complexité

Le problème est un Problème d'Optimisation Combinatoire (POC). La complexité est non polynomiale et c'est un PE associé NP-complet. Pour un ensemble de n points, on a $\frac{(n-1)!}{2}$ possibilités, c'est donc une complexité factorielle en n.

Nous avons 22 noeuds pour le problème étudié. Cependant, nous avons 5 adresses et nous en calculons les combinaisons grâce au graphe 2. Nous obtenons 12 chemins possibles. Il est donc aisé de comparer et de trouver le chemin idéal. Si, à contrario, nous avions le double d'adresses, nous aurions une complexité de 2520, ce qui montre la croissance exponentielle de ce type de problème.

3 Etape 2 : L'algorithme

3.1 Question 1 : Etat de l'art et sources

Etat de l'art

- Problème du voyageur de commerce - Wikipedia
- https://interstices.info/le-probleme-du-voyageur-de-commerce/
- https://math.unice.fr/~pauly/Little.pdf

Algorithmes

- https://rosettacode.org/wiki/Dijkstra's_algorithm#C
- https://www.programiz.com/dsa/dijkstra-algorithm
- https://en.wikipedia.org/wiki/Branch_and_bound
- https://www.jot.fm/issues/issue_2005_01/article5.pdf

3.2 Question 2 : Pseudo code

3.2.1 Pseudo Code général

```
1  g <- graphe du probleme
2  adresses <- noeuds de g étant des adresses
3  t <- nouveau graphe transformé à partir de Dijkstra
4
5  Pour chaque a dans adresses
6    dist, prev <- dijkstra(g, a)
7    Pour chaque b dans adresses
8      Si b != a et NON t.existeArc(a, b)
9        t.ajouterArc(a, b, dist[b])
10
11 meilleurChemin <- brute force sur t pour trouver le chemin hamiltonien
    le plus court
12
13
14
15
16
```

3.2.2 Pseudo Code Brute force

```
fonction bruteForceTSP(Graphe G, Sommet S): Liste de Sommet

  coutMini <- INFINI
  meilleurChemin <- Liste de sommet vide
  cheminsPossibles <- toutes les combinaisons de chemins possibles

  Pour chaque chemin dans cheminsPossibles Faire
    cout <- calcul du cout total du chemin
    Si cout < coutMini Alors
      coutMini <- cout
      meilleurChemin <- chemin

  Retourner meilleurChemin
```

3.2.3 Pseudo Code Djikstra

```
1  function dijkstra(G, S, metroEnPanne):
2
3    P <- Liste de sommet vide    /*liste des noeuds déjà explorés*/
4    Q <- Liste de sommet vide    /*liste des noeuds à explorer*/
5
6    Pour chaque sommet V dans G
7      distance[V] <- infini
8      previous[V] <- NULL
9      Si V != S Alors
10       ajouter V à Q
11   distance[S] <- 0
12
13
14   Tant que Q est NON VIDE
15     U <- noeud pas encore exploré le plus proche de S
16     retirer U de Q
17     ajouter U à P
18     Pour chaque sommet V dans Q
19       Si g.existeArc(U, V) Alors     /*Si V est voisin de U*/
20         Si NON metroEnPanne OU V PAS de type [métro] Alors  /*Si le métro
     n'est pas en panne ou si le noeud n'est pas de type métro*/
21           Si V PAS de type [adresse] OU V = S Alors    /*Si V n'est pas une
     adresse ou si V est le sommet de départ*/
22             tempDistance <- distance[U] + g.valeurArc(U, V) /*On calcul la
     distance entre le voisin et U*/
23             Si tempDistance < distance[V] Alors /*Si la distance est infé
     rieur à la précédente on met à jour*/
24               distance[V] <- tempDistance
25               previous[V] <- U
26
27   Retourner distance[], previous[]
28
29
30
31
32
```

4 Etape 3 : L'implémentation

4.1 Le graphe utilisé par le programme

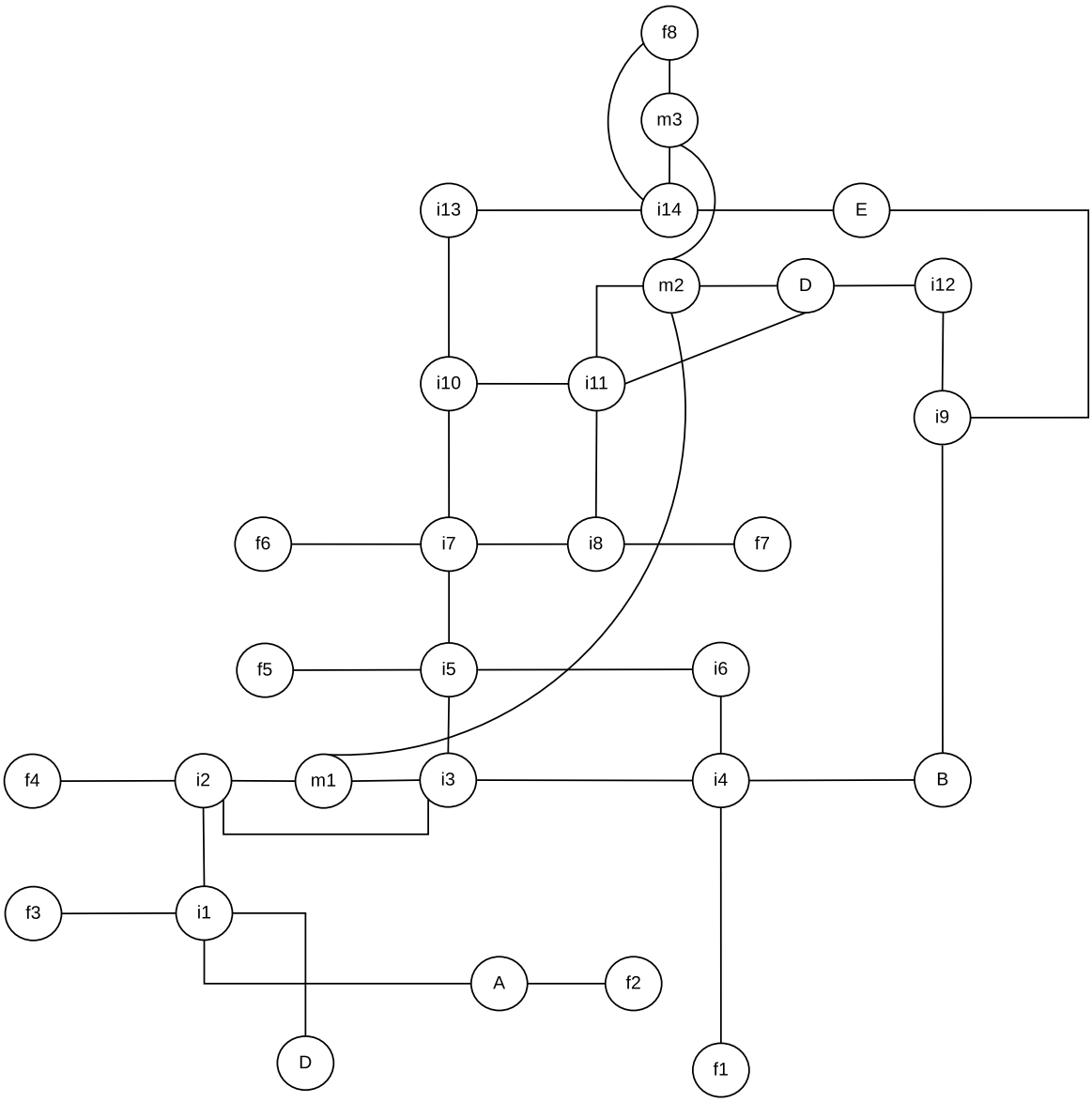


FIGURE 3 – Représentation sous forme de graphe du problème que la machine interprête

4.2 Le résultat

FIGURE 4 – Résultat du programme avec métro

5 Etape 4 : L'adaptation

5.1 Question 1 : Métro en grève ?

Si le métro est en grève, il faut pouvoir retirer les noeuds correspondant au métro ainsi que les arêtes. C'est pour cette raison que nous les avons modélisés différemment.

5.2 Question 2 : Conséquences sur l'algorithme

Pour que l'algorithme prenne en compte cette variable, il suffit de différencier les noeuds à l'avance et grâce à une simple condition sur le type de noeud, on peut exclure le métro des combinaisons.

5.3 Question 3 : Implémentation

```
------------------------------ SANS MÉTRO ------------------------------
Calcul d'un des meilleurs chemins sans métro...
Ordre des adresses à visiter: [D, A, B, E, C, D]
Le coût du chemin s'élève à: 20
Chemin à suivre: D - i1 - A - i1 - i2 - i3 - i4 - B - i9 - E - i9 - i12 - C - i11 - i8 - i7 - i5 - i3 - i2 - i1 - D
```

FIGURE 5 – Résultat du programme sans métro

6 Conclusion

Notre solution est simple et rapide et répond au besoin du problème énoncé. Elle n'est pas optimale pour les problèmes de grande envergure puisque nous utilisons une méthode gloutonne. Cependant, nos recherches nous ont amenés à réfléchir sur la méthode du Branch & Bound, aussi appelée Séparation et Evaluation. Elle permet d'optimiser le problème en évitant de calculer des coûts dont on peut se douter à l'avance qu'ils sont trop élevés, et ainsi retirer des calculs. La méthode était assez complexe et, dans notre cas précis, inutile puisque le nombre d'adresses est petit. Nous aurons donc pu constater la complexité du problème du voyageur de commerce et les pistes de solutions qui existent. Ainsi, il se résoud facilement, mais à condition d'avoir peu de noeuds, ce qui est rarement le cas dans la réalité...