Compte Rendu

MU4IN200

Projet de Modélisation, optimisation, graphes, programmation linéaire

Binôme :

Qingyuan Yao
Jules Cassan

Encadrant :

Euripidis Bampis

Sommaire

1 Introduction

L'objectif de ce projet est d'étudier différents chemins dans un type de graphe particulier. Nos graphes ne pourrons pas avoir de cycle et la valeur des arcs ne peux pas être négative. Les arcs aurons deux paramètres, un poids qui représenteras la durée du trajet entre deux sommets et une date représentant le moment exact où l'arc est "ouvert" au passage. Dans ce projet nous allons étudier différentes manières de traverser notre graphe ainsi que les algorithmes de parcours qui les accompagnes.

2 Question 1 : Assertions

2.1 Assertion 1

Un sous-chemin préfixe d'un chemin d'arrivée au plus tôt peut ne pas être un chemin d'arrivé au plus tôt

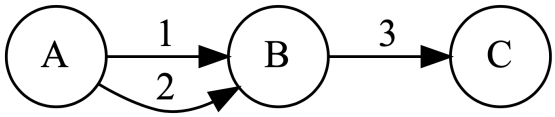

FIGURE 1 – Exemple pour assertion 1

Comme vu dans Figure 1, **A-2->B-3->C** est l'un des chemins d'arrivée au plus tôt de A à C, mais **A-2->B** n'est pas le chemin d'arrivée au plus tôt de A à B (**A-1->B** est plus tôt)

2.2 Assertion 2

Un sous-chemin postfixe d'un chemin de départ au plus tard peut ne pas être un chemin de départ au plus tard.

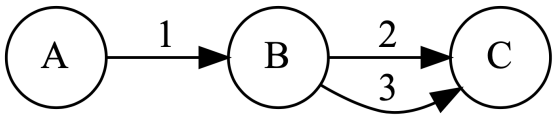

FIGURE 2 – Exemple pour assertion 2

Comme vu dans Figure 2, **A-1->B-2->C** est l'un des chemins de départ au plus tard de A à C, mais **B-2->C** n'est pas le chemin de départ au plus tard de B à C (**B-3->C** est plus tard)

2.3 Assertion 3

Un sous-chemin d'un chemin le plus rapide peut ne pas ˆetre un chemin le plus rapide.

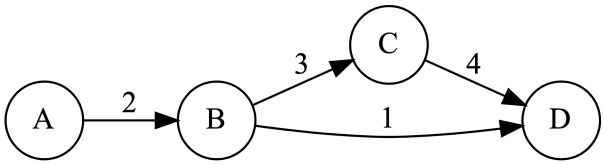

FIGURE 3 – Exemple pour assertion 3 et assertion 4

Comme vu dans Figure 3, **A-2->B-3->C-4->D** est le chemin le plus rapide de A à D, mais **B-3->C-4->D** n'est pas le chemin de départ au plus tard de B à D (**B-1->D** est plus rapide)

2.4 Assertion 4

Un sous-chemin d'un plus court chemin peut ne pas être un plus court chemin.

Comme vu dans Figure 3, **A-2->B-3->C-4->D** est le plus court chemin de A à D, mais **B-3->C-4->D** n'est pas le plus court chemin de B à D (**B-1->D** est plus court)

3 Question 2&3 : Algorithmes et Complexités

3.1 Fonctions utiles

Voici deux fonctions qui vont être utiliser dans les 4 algorithmes suivants, et qui contribuent beaucoup à leurs complexités

3.1.1 Bellman

Nous utiliseront l'algorithme de Bellman dans tous nos algorithmes non seulement pour calculer les chemins les plus courts, mais en même temps pour vérifier s'il existe un chemin entre deux sommet.
L'algorithme retourne 2 listes :

- Une liste *distance* qui contient les distances du sommets de départ à chaque sommets prenant les chemins plus courts
- Une liste *chemin* qui contient pour chaque sommet, le sommet d'avant en prenant les chemins plus courts depuis le sommet de départ, grâce à cela, tous les chemins sont retraçables depuis cette liste

Nous vérifieront aussi s'il existe un circuit absorbant

Complexité :

$$\begin{aligned} C(bellman) &= C(algo\_bellman) + C(verification\_circuit) \\ &= O(nb\_sommets^2) + O(nb\_sommets) \\ &= O(nb\_sommets^2) \end{aligned}$$

3.1.2 tracer_Chemin

On part de la fin, lors qu'on n'atteint pas le début(None), on continue le while On garde qu'une seule fois le sommet, c'est-à-dire (A,1)->(A,2) = A Si le début n'est pas la fin (None empêche le dernier sommet a être ajouté), on le rajoute manuellement

Complexité :

$$\begin{aligned} C(tracer\_Chemin) &= C(parcourir\_liste\_chemin) \\ &= O(nb\_sommets) \end{aligned}$$

3.2 Type I : Chemin d'arrivée au plus tôt

Pré-condition : l'ordre des sommets est déjà trié (dans type 1,2, nous bénéficions de l'indifférence du sommet de départ ou d'arrivée, et aussi le tri des sommets pour accélérer les algorithmes)

1. Nous récupérons les sommets de départ et de l'arrivée à traiter (nous ne gardons que le premier sommet de départ, parce qu'on peut en aller partout et nous ne distinguons pas les sommets de départ dans ce type)
2. Nous lançons l'algorithme de Bellman depuis le seul sommet de départ (dans type 1,2 3, Bellman ne sert qu'à la vérification de l'existence d'un chemin entre les sommets de départ et d'arrivée)
3. Depuis le premier sommet d'arrivées, nous prenons le premier sommet d'arrivée atteignable depuis le sommet de départ, puisque les sommets sont triés, il sera forcement le sommet d'arrivée dont le jour est plus tôt

Complexité :

$$\begin{aligned} C(type1) &= C(get\_departs\_arrivees) + C(bellman) + C(chercher\_atteignable) \\ &\quad + C(tracer\_Chemin) \\ &= O(nb\_sommets) + O(nb\_sommets^2) + O(nb\_sommets) + O(nb\_sommets) \\ &= O(nb\_sommets^2) \end{aligned}$$

3.3 Type II : Chemin de départ au plus tard

Pré-condition : l'ordre des sommets sont déjà trié

1. Nous récupérons les sommets de départ et de l'arrivée à traiter (nous ne gardons que le dernier sommet d'arrivée, parce qu'on peut y arriver de n'importe quel sommet de départ et nous ne distinguons pas les sommets d'arrivée dans ce type)
2. Nous inversons la liste des sommets de départ parce qu'on cherche le sommet avec la plus grande valeur
3. Pour chaque sommet de départ, nous lançons l'algorithme de Bellman depuis le sommet
4. Dès qu'un chemin est trouvé, nous arrêtons la boucle et nous prenons le sommet de départ au courant, puisque les sommets sont triés dans le sens inverse, il sera forcement le départ le plus tard.

Complexité :

$$\begin{aligned} C(type2) &= C(get\_departs\_arrivees) + C(inverser\_liste) + nb\_sommets \\ &\quad * (C(bellman) + C(chercher\_atteignable)) + C(tracer\_Chemin) \\ &= O(nb\_sommets) + nb\_sommets * (O(nb\_sommets^2) + O(nb\_sommets)) \\ &\quad + O(nb\_sommets) \\ &= O(nb\_sommets^3) \end{aligned}$$

3.4 Type III : Chemin le plus rapide

1. Nous récupérons les sommets de départ et de l'arrivée à traiter (dans ce type, nous gardons tous les sommets, parce que nous nous intéressons au départ ET à l'arrivée)
2. Nous inversons la liste des sommets de départ parce qu'on cherche le sommet avec la plus grande valeur
3. Pour chaque sommet de départ, nous lançons l'algorithme de Bellman depuis le sommet. Nous sauvegardons les durées des tous les sommets de départ avec tous les sommets d'arrivée s'il existe un chemin pour atteindre
4. Nous cherchons la durée la plus petite dans notre sauvegarde, et nous retournons le sommet de départ et le sommet d'arrivée correspondants

Complexité :

$$\begin{aligned} C(type3) &= C(get\_departs\_arrivees) + C(inverser\_liste) + nb\_sommets \\ &\quad * (C(bellman) + C(add\_to\_list)) + C(chercher\_min) \\ &\quad + C(tracer\_Chemin) \\ &= O(nb\_sommets) + nb\_sommets * (O(nb\_sommets^2) + O(nb\_sommets)) \\ &\quad + O(nb\_sommets^2) + O(nb\_sommets) \\ &= O(nb\_sommets^3) \end{aligned}$$

3.5 Type IV : Plus court chemin

1. Nous ne récupérons que le premier sommet de départ et le dernier sommet d'arrivée (puisque la distance entre les départs et les arrivées sont 0)
2. Nous lançons l'algorithme de Bellman depuis le sommet de départ récupéré (grâce à Bellman, en prenant le premier départ de la dernière arrivée, nous obtiendrons le plus court chemin qui considère tous les sommets)

Complexité :

$$\begin{aligned} C(type4) &= C(get\_departs\_arrivees) + C(bellman) + C(tracer\_Chemin) \\ &= O(nb\_sommets) + O(nb\_sommets^2) + O(nb\_sommets) \\ &= O(nb\_sommets^2) \end{aligned}$$

4 Question 4 : Programme

4.1 Affichage.py

Affichage.py contient les fonctions d'affichage de graphe, grâce à l'extension graphviz

4.2 Generate.py

Generate.py contient les fonctions de génération aléatoire de graphes

4.3 Algo.py

Algo.py contient les fonctions des algorithmes vus dans Section 3

4.4 Programmation_Lineaire.py

Programmation_Lineaire.py contient les fonctions de résolution du plus court chemin d'un graphe en PL (cf. Section 5)

4.5 Main.py

Main.py contient les fonctions d'interface et aussi de lecture des graphes

5 Question 5 : Modélisation

Pour modéliser notre problème de plus court chemin en programme linéaire, nous avons tout d'abord convertis un graphe en programme linéaire. Ses sommets sont devenus les variables de décisions et ses arcs les contraintes de notre programme linéaire. Pour exemple, le sommet 1 et le sommet 2 sont devenus les variables x1 et x2 et l'arc qui réalisais la liaison entre les deux ce traduit par la contrainte x2 - x1 <= 1. On réaliste cette opération pour tout notre graphe. Il ne nous reste plus qu'a exprimer le problème du plus court chemin entre deux point avec la fonction objectif. Ici, il ne s'agissait pas de minimiser la distance entre Xstart et Xend mais justement de la maximiser. En effet, en maximisant la distance, cela permet de mettre en tension l'ensemble du programme linéaire pour que le problème ne sois pas non borné.

Dans notre projet, c'est la fonction conversion_PL(graphe,start,end) qui convertis les paramètres d'entrée en un programme linéaire et le solveur guroby ce charge ensuite de calculer la valeur de distance la plus courte et nous permet de retracer le chemin le plus court avec les variables mise en tension (leur valeur est différente de 0).

6 Question 6 : Tests

6.1 Influence du nombre de Sommet

Le nombre de sommet correspond au nombre de variables de décisions dans notre programme linéaire. Cela allonge de plus en plus le temps de calcul du chemin du

plus court chemin.

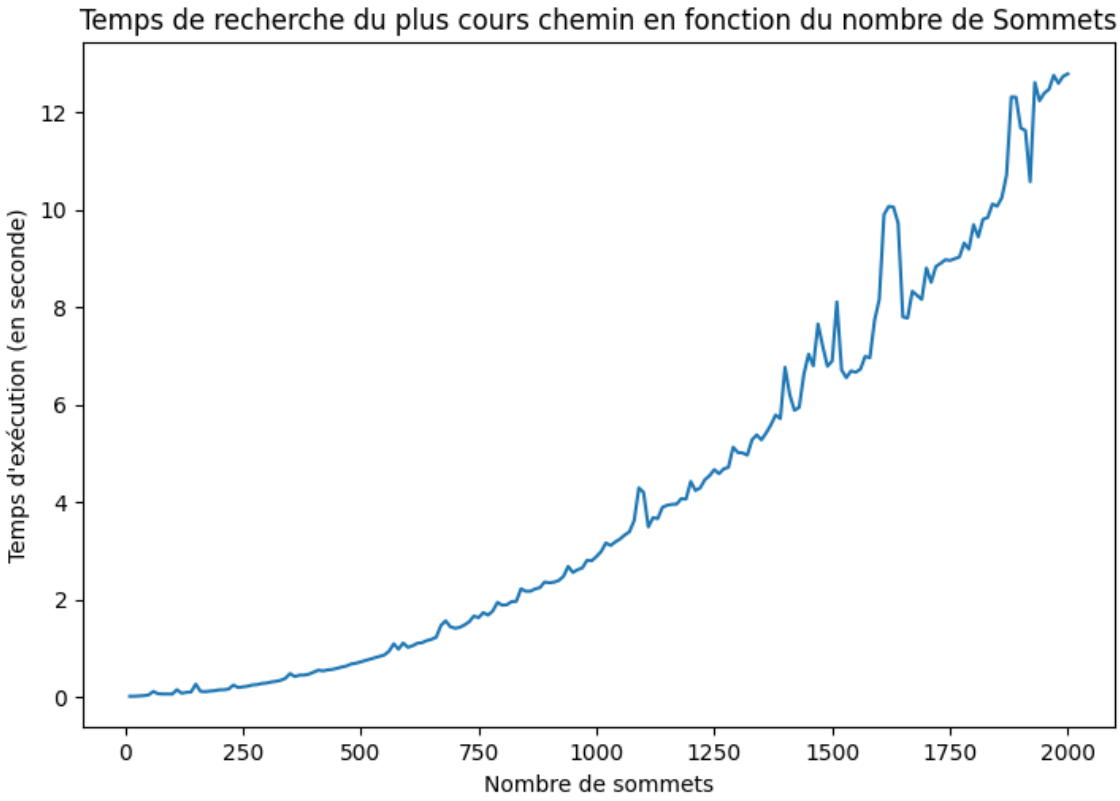

FIGURE 4 – Graphique de temps d'exécution pour les sommets

Comme on peux le voir sur sur le graphique le temps augmente avec ce qui semble être un facteur multiplicatif lié au nombre de sommet .

6.2 Influence du nombre d'Arc

Le nombre d'arc correspond lui au nombre de contraintes de notre programme linéaire ce qui prend plus de temps a calculer comparé à l'ajout d'un sommet. Comme pour les sommet, l'ajout d'arc augmente avec un facteur multiplicatif plus important ici.

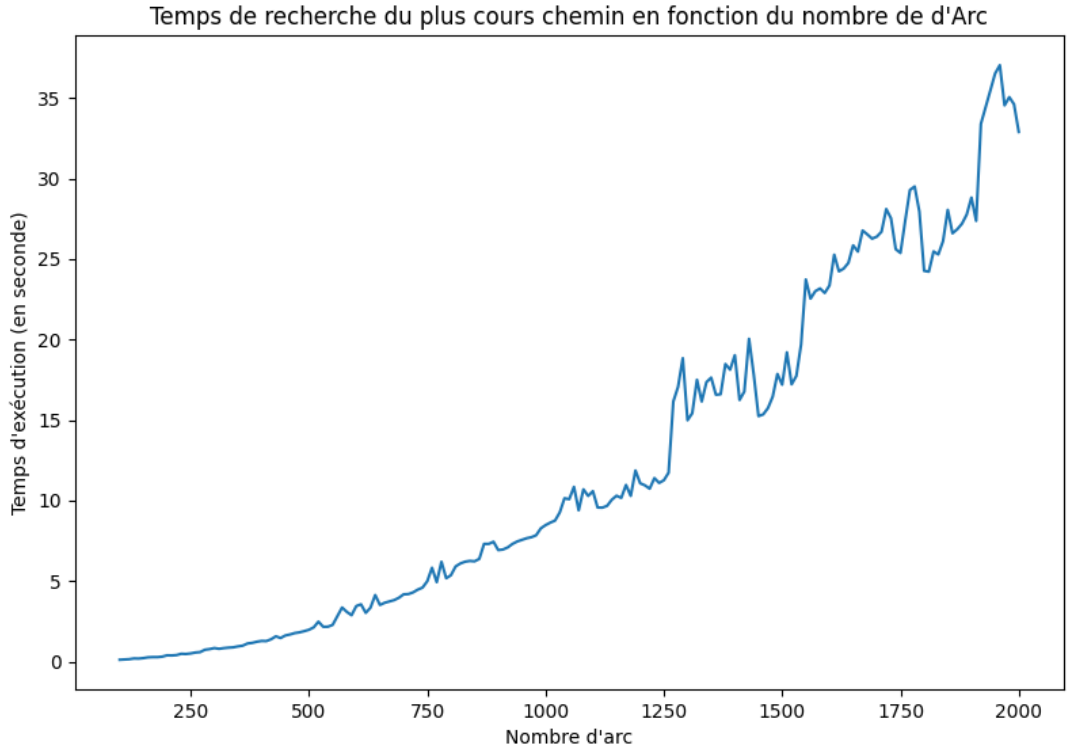

FIGURE 5 – Graphique de temps d'exécution pour les arcs

Les test ont été effectué avec les méthodes test_arcs_guroby() et test_sommets_guroby qui effectue le calcul du plus court chemin d'un graphe généré avec un nombre d'arc et de sommet donnée. Les temps d'exécution du calcul sont écrit dans les fichier test_arcs.txt et test_sommets.txt pour pouvoir être utilisé plus tard.

On peux conclure a l'issue de ces tests que l'ajout de nouveaux arcs impact le plus le temps de calcul. Ce qui peux nous amenez dans certains cas a calculer le dual de notre programme linéaire afin de convertir nos variables de décision en contraintes et vis versa afin de gagner en rapidité de calcul.

7 Question 7 : Comparaison

Les algorithmes de calculs du plus court chemin sont bien différents à la fois dans leur performances et dans implémentation. En effet, comme le montre le graphique suivant, la résolution par programme linéraires est largement plus efficace que notre algorithme. Cela s'explique par notre volonté de passer d'un graphe initial à un graphe modifier/simplifier comme expliquer avant. Et avec une implémentation comme la notre, il est peu efficace de parcourir plusieurs fois les deux graphique pour

retrouver les identifiant correspondant à chaque recherche de plus court chemin

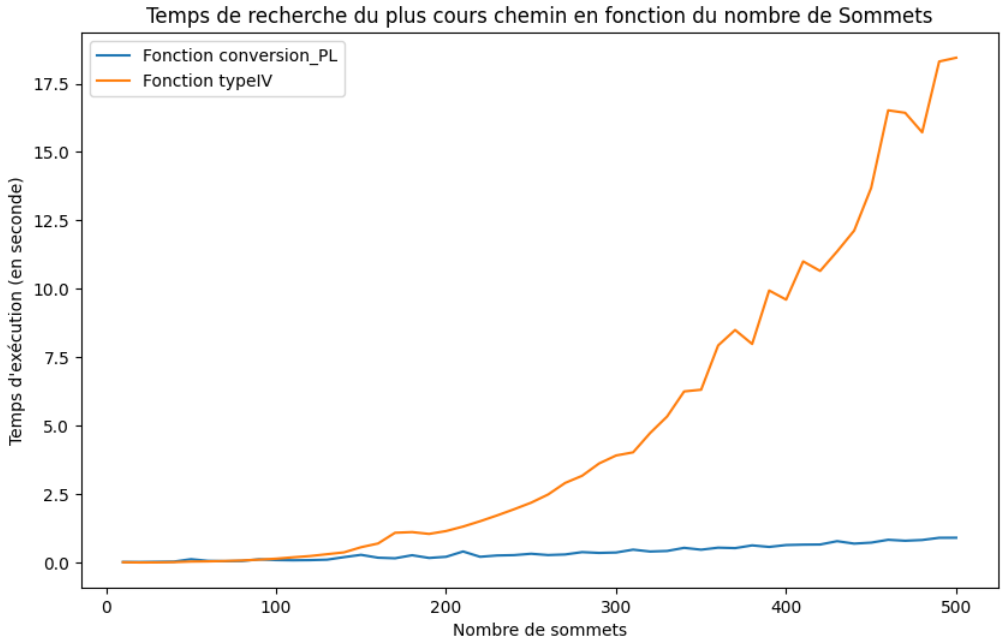

FIGURE 6 – Graphique de comparatif du temps d'exécution des fonction type4 et conversion_PL

8 Conclusion

En conclusion, nous avons pu à travers ce projet comprendre qu'il est possible de traduire un problème avec plusieurs contraintes et qu'en modifiant la structure du problème sans en changer la nature on pouvais aboutir au même résultat en simplifiant le travail de calcul.

Annexes

A Modèles Graphe

Nous avons utilisé le graphe présent dans le sujet dans le fichier graphe.txt, ainsi qu'un graphe_complexe que nous avons construit qui est plus complexe et plus grand. Et les graphes d'exemple pour les assertions de question 1 (cf. Section 2)

B Modèles PL

Pour la modélisation en PL, nous avons référer un document[2] qui explique le lien entre le problème du plus court chemin avec le dual en PL.

Programme primal

$$\text{minimiser } \sum_{e \in A} x_e poids(e)$$
$$\begin{cases} \sum_{e \text{ partant de } s} x_e - \sum_{e \text{ sortant de } s} x_e & = 1 \\ \sum_{e \text{ partant de } t} x_e - \sum_{e \text{ sortant de } t} x_e & = -1 \\ \sum_{e \text{ partant de } u} x_e - \sum_{e \text{ sortant de } u} x_e & = 0 \text{ pour tout sommet } u \notin \{s,t\} \\ x_e \geq 0 \text{ pour tout arc } e \end{cases}$$

Programme dual

$$\text{maximiser } d_t - d_s$$
$$\begin{cases} d_v - d_u \leqslant poids(u,v) \text{ pour tout arc } u \to v \\ d_u \in \mathbb{R} \text{ pour tout sommet } u \end{cases}$$

FIGURE 7 – Modèles plus court chemin en Primal-Dual

C Tests réalisées

Les fonctions utilisées pour les tests sont stockées dans Main.py (cf. Section 4.5), les resultats sont stockés dans les fichiers test_arcs.txt et test_sommets.txt

D Repertoire git [3]

Nous fournissons également tous les fichiers concernés, y compris les programmes en python (cf. Section 4), les graphes (cf. Section A), les résultats des tests, et les graphiques (cf. Section 6)

Références

[1] graphviz : `https://www.graphviz.org`

[2] Modèles Primal-Dual : `http://people.irisa.fr/Francois.Schwarzentruber/algo2/04dualite.pdf`

[3] Repertoire git : `https://github.com/MRVNY/MOGPL.git`