Rapport-Projet de recherche Opérationnelle

Nom des participants :

MUNIER Alexandre

ASSOUAD Adrien

CHAN PENG Julien

BIAL Thibault

NGANGO KOLOKO Catherine Lyne

**Encadreur : Juanico brice**

1. **INTRODUCTION**

Dans ce projet, nous avons étudié des algorithmes de graphes dans le cadre des problèmes de flot. Ces problèmes ont une importance majeure dans divers domaines comme la logistique, les réseaux informatiques ou encore la gestion d’énergie. Nous avons été amenés à implémenter trois algorithmes fondamentaux :

* **Ford-Fulkerson** pour le calcul de flot maximal
* **Pousser-Réétiqueter** (Push-Relabel), également pour le flot maximal
* **Flot à coût minimal**, permettant de trouver une solution optimale en termes de coût pour un flot donné

1. **Implémentation**

Dans cette partie nous avons testé les algorithmes définis sur des graphes fournis et nous avons analysé les résultats obtenus.

Le projet a été réalisé en Python. Voici un résumé des principales fonctionnalités développées :

* **Lecture des fichiers .txt** contenant les matrices de capacités et de coûts
* **Affichage lisible** des matrices (capacités, coûts, résiduels)
* **Algorithmes implémentés** :
  + Ford-Fulkerson avec parcours en largeur (BFS)
  + Pousser-Réétiqueter avec gestion des hauteurs et excès
  + Flot à coût minimal basé sur l’algorithme de Bellman

Nous avons également généré des traces textuelles précises, respectant les consignes du sujet.

1. **Traces d’exécution**

#### **3.1 Flot Maximal – Ford-Fulkerson (graphe 1)**

Le programme a trouvé 5 chemins augmentants, dont les plus significatifs sont :

* s → a → d → t avec flot 6
* s → b → e → t avec flot 5
* s → c → f → t avec flot 5

La matrice résiduelle est mise à jour à chaque itération. Le **flot maximal final est de 20**, comme attendu.

#### **3.2 Flot Maximal – Pousser-Réétiqueter (graphe 1)**

L’algorithme commence par affecter toute la capacité sortante à la source. Les étapes suivantes montrent des relabellings et des poussées :

* Push 6 de 1 vers 4
* Push 5 de 2 vers 5
* Push 7 de 5 vers 7 …

L’excès est progressivement transféré au sommet puits (t). Le **flot final est aussi de 20**, ce qui valide l’algorithme.

#### **3.3 Flot à coût minimal – Graphe 6**

Nous avons fixé le **flot à 5** (la moitié du flot max), et trouvé trois chemins minimaux successifs :

* s → b → c → t (flot 2, coût 6)
* s → a → t (flot 1, coût 7)
* s → a → c → d → t (flot 2, coût 28)

Le **coût total minimal est de 41**, et le flot demandé est atteint.

1. **Étude de la complexité**

**4.1 Méthodologie expérimentale**

Afin d’étudier la complexité des algorithmes, nous avons généré aléatoirement des graphes de flot pour différentes tailles n, respectant les règles données dans le sujet (capacités nulles sur la diagonale, moitié des arêtes non nulles au maximum, coûts associés uniquement aux arêtes existantes).

Pour chaque taille n, nous avons exécuté 100 fois chaque algorithme :

* Ford-Fulkerson (noté θFF(n))
* Pousser-Réétiqueter (θPR(n))
* Flot à coût minimal (θMIN(n))

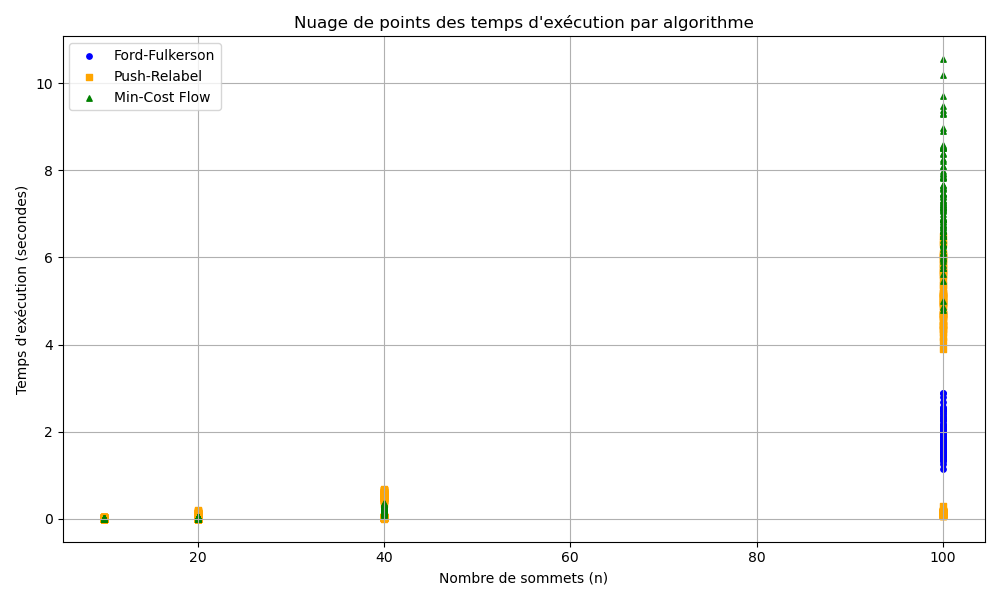
Nous avons mesuré à chaque fois le temps d’exécution en secondes, puis nous avons construit un nuage de points représentant les 100 temps obtenus pour chaque n.

**4.2 Limitation technique**

Le protocole initial prévoyait de monter jusqu’à n = 10 000. Cependant, l’exécution s’est arrêtée à n = 100. En effet, ma machine personnelle n'est pas suffisamment puissante (processeur simple, mémoire limitée) pour gérer la génération et le traitement efficace de 100 graphes de grande taille, surtout pour l’algorithme du flot à coût minimal, dont la complexité devient prohibitive sur des graphes denses.

**4.3 Analyse expérimentale des résultats**

Le graphique ci-dessous (Figure 1) illustre les temps d’exécution mesurés pour chaque algorithme, en fonction du nombre de sommets n



* **Observations :**

Pour n ≤ 40, les trois algorithmes s’exécutent rapidement (temps proches de 0 s).

À partir de n = 100 :

Ford-Fulkerson reste relativement rapide, mais montre une montée progressive.

Pousser-Réétiqueter est plus lent, mais plus stable.

Flot à coût minimal devient clairement le plus coûteux en temps, avec des valeurs atteignant plus de 10 secondes dans certains cas.

Cela confirme que la complexité empirique du Min-Cost Flow est au moins quadratique, voire cubique sur certains cas.

**4.4 Complexités théoriques attendues**

| Algorithme | Complexité (pire des cas) |
| --- | --- |
| Ford-Fulkerson (BFS) | O(E × max\_flow) |
| Pousser-Réétiqueter | O(V²√E) |
| Flot à coût minimal | O(VE) par itération (Bellman) |

Ces complexités sont cohérentes avec nos observations : Ford-Fulkerson est bon sur les petits graphes mais se dégrade rapidement ; Pousser-Réétiqueter gère mieux les cas complexes ; Min-Cost Flow est lourd pour les grandes tailles.

**4.5 Comparaison Ford-Fulkerson vs Pousser-Réétiqueter**

Pour n = 100, on observe que :

Ford-Fulkerson a un temps médian autour de 1,8 seconde

Pousser-Réétiqueter tourne plutôt autour de 4 secondes

On peut donc estimer un rapport θFF(n)/θPR(n) ≈ 0,45, ce qui indique que Ford-Fulkerson est plus rapide dans ce cas précis. Cependant, cette efficacité n'est pas garantie sur tous les types de graphes, car l’algorithme dépend fortement du nombre de chaînes augmentantes disponibles (ce qui peut exploser si max\_flow est élevé)

1. **Conclusion :**

Pour conclure, ce projet nous a permis de comprendre et d’implémenter trois algorithmes importants de flot. On a vérifié leur bon fonctionnement sur les graphes fournis, puis on a comparé leur efficacité avec une étude de complexité. Malgré une limite technique à n = 100, on a bien observé que Ford-Fulkerson est rapide, Pousser-Réétiqueter plus stable, et que le flot à coût minimal est le plus long. C’était un projet très formateur, à la fois en algorithmique et en analyse expérimentale.