INFO-F310: Maximum flow

RACHIK Hilal 520550
RACHIK Nadim 480729

avril 2023

Contents

1 Introduction

Le problème de flot maximum consiste a envoyer un maximum de ressources d'un noeud source (s) a un noeud puits (t) dans un graphe $G = (V, A)$ ou chaque arc $ij \in A$ a une capacité maximum c_{ij}.
Ce problème peut être modélisé et résolu a l'aide d'un programme linéaire mais également par la méthode des chemins augmentant.
Dans ce rapport, nous allons décrire et comparer les performances de ces 2 méthodes de résolution (tests sur **Windows**).

2 Méthodes

2.1 Linear program

$$\begin{aligned} &\max && v \\ &\text{s.c.} && \sum_{j:(i,j)\in A} f_{ij} - \sum_{j:(j,i)\in A} f_{ji} = \begin{cases} v & i = s, \\ -v & i = t, \\ 0 & \text{sinon}, \end{cases} && i \in N, \\ & && 0 \le f_{ij} \le u_{ij} && (i,j) \in A. \end{aligned}$$

Figure 1: Max Flow LP formulation [2]

- Variables:

  $f_{i,j}$ = Flot envoyée sur l'arrête (i, j)

- Contraintes/Objectif:

  il y a 2 types de contraintes, les contraintes de capacité qui assurent que la capacité de chaque arrête soit respectée (intuitif), et les contraintes de conservation qui assurent que le flow est conservé dans le graphe.

  contraintes de conservation:

  - Pour tout noeuds standards (non source/puits), la somme des flows sortants = somme des flows entrants. Cela assure la conservation du flow a travers le graphe.
  - La source émet V flow en plus qu'elle en reçoit.
  - Le puits reçois V flow en plus qu'il en émet.

  Objectif: V, on maximise la quantité que la source peut émettre "en plus" et que le puits peut recevoir.

- Complexité (génération contraintes):

  - capacité: $O(|E|)$ (une contrainte par arrête)
  - conservation: $O(|V| * 2|E|) = O(|V| * |E|)$ dans le pire des cas on parcours chaque arrête 2 fois par noeud)

  Donc, la génération de contraintes est en: $O(|V| * |E|)$

2.2 Augmenting path

L algorithme utilise est l algorithm d'Edmonds–Karp [3], une implémentation de l algorithme de Ford Fulkerson en utilisant un parcours BFS pour la recherche de chemins augmentant (chemins de taille minimale).

2.2.1 Recherche de chemin

Comme explique ci dessus, la recherche de chemin augmentant est réaliser a l aide d un parcours bfs, cela permet de trouver le chemin augmentant le plus court possible a chaque itération. Le chemin est récupérer a travers une liste de prédécesseur qui est parcourue a partir du noeud puits (a l envers).

- Puits **marqué**: un chemin augmentant a été trouvé (t marqué), donc fin de recherche (augmentation sur le chemin trouve).
- Puits **non marqué**: aucun chemin augmentant n a été trouvé ($queue = \emptyset$), donc fin de l'algorithme.

2.2.2 Augmentation

```
# Phase d'augmentation

max_flow = 0
Tant que bfs():

    path_flow = +inf
    v = t (puits)

    # flot max du chemin
    Tant que v != s (source):
        path_flow = min(path_flow, graph[parent[v]][v])
        v = parent[v]

    max_flow += path_flow

    v = t (puits)
    Tant que v != s:
        u = parent[v]
        graph[u][v] -= path_flow  # augmantation sur le chemin
        v = parent[v]

return max_flow
```

Figure 2: Pseudo-code: phase d'augmentation

La phase d'augmentation augmente le flot le long du chemin augmentant trouvé a la dernière phase de recherche. La première étape sert a déterminer **max_flow** le flot maximum supplémentaire que le chemin peut recevoir. La deuxième étape consiste a mettre a jour le graphe résiduel, la soustraction est contre intuitive car on parle d'augmentation, mais en pratique on diminue le flot disponible sur chaque arrête, la capacité est consommée.

L algorithme alterne entre phase de recherche et phase d augmentation, jusqu a ce qu il n y ai plus de chemins augmentant, en augmentant le flot maximum a chaque itération par le flot du chemin trouvé.

- Complexité:
  - Recherche: $O(|V| + |E|)$ (parcours bfs)
  - Augmentation: $O(2 * |V|) = O(|V|)$ (les 2 boucles whiles internes parcourent le chemin trouvé qui dans le pire des cas est "hamiltonien" et passe par tout les noeuds)

  Dans le pire des cas le flot est augmenté $|V| * |E|$ fois [1], donc le nombre d iteration de la phase d augmentation est bornée par la même valeur.

  Donc, l'algorithme est en: $O(|E| * |V| * (|E| + |V|) \approx O(|V| * |E|^2)$

2.3 Minimum cut verif.

Pour vérifier que la solution est optimale, on peut comparer la valeur de la coupe minimum a la solution, l'**égalité** est requise. La coupe minimum est obtenue a partir du dernier marquage (les noeuds marqué lors de la dernière itération interrompue par absence de chemins augmentant)

Input : cut C, graph G = (V,E)
Output: Value of C

```
Value of C ← 0;
for (i, j) ∈ E, (i ∈ cut C) do
    if j ∉ cut C then
        Value of C ← Value of C + c_{i,j};
    end
end
```

Algorithm 1: Value of a graph cut

Explications: La coupe minimum d'un graphe est une partition des sommets en deux ensembles (côté source et côté puits), telle que la somme des capacités des arêtes traversant la partition soit minimale. Donc, si la valeur du flux maximum est égale a la valeur de la coupe minimum, cela signifie que toute la capacité disponible pour atteindre le cote puits est utilisée, donc on ne peut plus augmenter le flux maximum.

3 Résultats

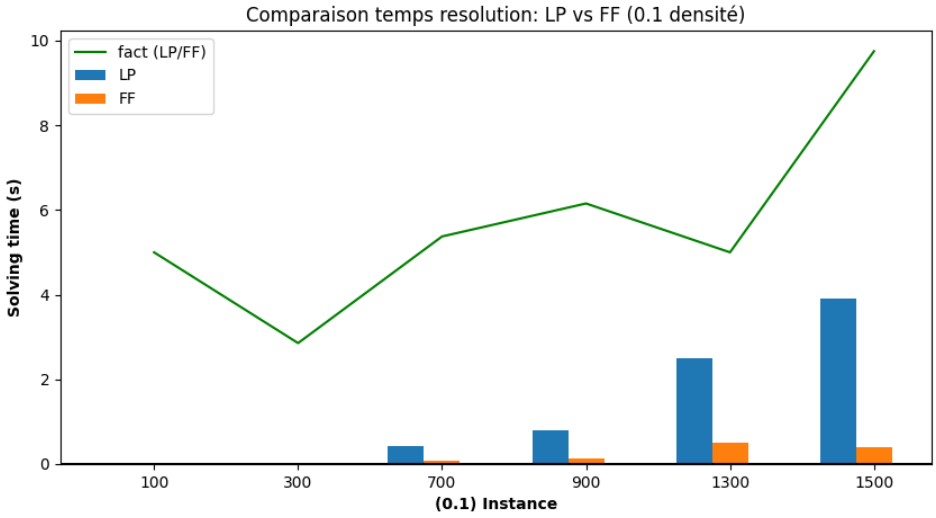
Gestion des arcs doubles: on considère la dernière capacité enregistrée

Temps LP: le calcul des temps de résolution pour la méthode LP est: t(génération model) + t(résolution model). (résolution model = "time used" du rapport de résolution donné en console par glpk)

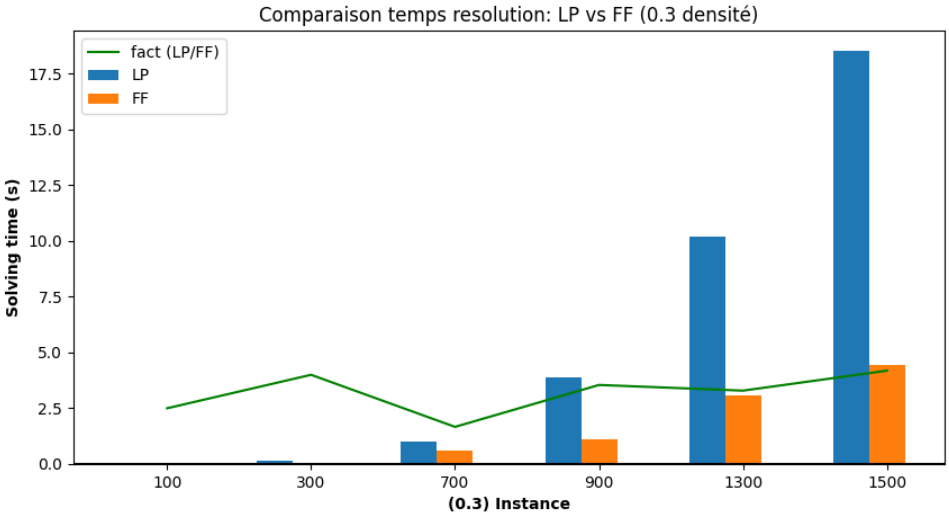
Graph parse: Les 2 méthodes travaillent avec la même classe Graph, donc le temps de traitement du fichier txt en un objet Graph est ignoré (en plus d être négligeable)

N	P	FF (s)	LP (s)	Max flow
100	0.1	0.001	0.005	72
	0.2	0.002	0.0	88
	0.3	0.002	0.005	151
300	0.1	0.007	0.02	167
	0.2	0.02	0.13	320
	0.3	0.04	0.16	478
500	0.1	0.04	0.16	244
	0.2	0.13	0.4	571
	0.3	0.18	0.73	748
700	0.1	0.08	0.43	361
	0.2	0.25	1.1	818
	0.3	0.6	1	1151
900	0.1	0.13	0.8	481
	0.2	0.45	2.4	939
	0.3	1.1	3.9	1499
1100	0.1	0.23	1.5	547
	0.2	0.75	4.5	1179
	0.3	2.2	6.8	1753
1300	0.1	0.5	2.5	613
	0.2	1.3	7.3	1440
	0.3	3.1	10.2	1999
1500	0.1	0.4	3.9	764
	0.2	1.6	10.4	1601
	0.3	4.42	18.5	2333

Table 1: Tableau des resultats et temps de résolution
(FF = Ford Fulkerson, LP = glpsol)



(a) $p = 0.1$



(b) $p = 0.3$

Figure 3: Graphes: comparaison temps de résolution

4 Discussion

Premièrement, on observe un avantage significatif de la méthode FF en terme de temps de résolution, en étant plus rapide sur toutes les instances testées.

La méthode LP et en particulier la résolution par le solveur glpk peut impliquer des étapes additionnelles qui peuvent augmenter les temps de résolutions. Tel que "Perturbing LP to avoid stalling" qui est une méthode de l'algorithme interne du solveur glpk, qui modifie la formulation pour améliorer la stabilité et éviter des problèmes de résolution important.

$|V|$ scalability

On peut comparer les comportements des temps de résolutions par rapport a l augmentation de la taille du graphe, en gardant la même densité ($|V| \nearrow, |E| \longrightarrow$).

- FF: L évolution est plus ou moins linéaire (résultats prévisibles).
- LP: L évolution se rapproche d un comportement exponentiel

$|E|$ scalability

On peut également comparer les comportements des temps de résolutions par rapport a l augmentation de la densité p du graphe, en gardant le même nombre de noeud ($|V| \longrightarrow, p \nearrow \Longleftrightarrow |E| \nearrow$).

Les calculs n'ont pas été faits, mais visuellement a partir du tableau des résultats on peut observer un meilleur comportement pour la méthode LP. Cela peut être expliqué par le degré de dépendance des 2 méthodes par rapport au nombre d arrêtes $|E|$, comme expliqué ci dessus le nombre de contraintes pour la méthode LP est linéaire en $|E|$ tandis que l algorithme FF est quadratique en $|E|$.

5 Conclusion

Nous avons analyser les comportements des deux méthodes différentes de résolution du problème de flot maximum, la méthode des chemins augmentant (FF) et celle du programme linéaire (LP). Nous avons observer et discuter leurs différences de comportement selon les caractéristiques des instances, principalement la taille et la densité du graphe.

Finalement, la méthode FF propose de meilleurs résultats a l'échelle des instances fournies, mais montre une limite car nous avons vu que l évolution par rapport au nombre d arrête était meilleur pour la méthode LP.

References

[1] Brilliant. Edmonds-karp algorithm, 2023.

[2] Bernard Fortz. Algorithmique 3 et recherche opérationnelle. Course materials, 2022.

[3] GeeksforGeeks. Ford-fulkerson algorithm for maximum flow problem, 2023.