1em 0pt

Département d'Informatique
Bachelor 3ème année de sciences informatiques

Algorithmique et recherches opérationnelles

Problème du Flot maximal

Enseignant : Bernard Fortz

par

Noe Bourgeois
et
Thomas Suau

Bachelor Sciences Informatiques Année 2022-2023

Table des matières

1 Introduction

1.1 Présentation du problème

Le problème de flot maximum est un problème d'optimisation en théorie des graphes, qui consiste à trouver le flot maximum qu'il est possible d'envoyer d'un nœud source à un nœud puits dans un graphe pondéré orienté où les arcs ont des capacités maximales. On en donne une expression ci-dessous 1.2.

Dans ce rapport, nous allons présenter deux méthodes pour résoudre le problème de flot maximum. La première méthode est la résolution à l'aide d'un programme linéaire avec le solver glpk, et la seconde méthode est la méthode des chemins augmentants implémentée en python3.

Dans notre problème les instances sont présentées sous la forme :
nodes n (nombre de noeuds)
source u (numéro de la source)
sink v (numéro du puit)
arcs a (nombre d'arcs)
i j c (présence d'un arc (i, j) et capacité $c_{i,j}$ associée)

1.2 Outils mathématiques

Soit $G = (A, N, C)$ un graphe, s sa source, t son puit et dont chaque arc (i, j) est pondéré par sa *capacité* $c_{i,j} \in C$.
La capacité pourrait être n'importe quel nombre réel positif mais dans ce cours on se concentre sur des capacités entières.

On appelle *flot* sur G tout vecteur $(f_{i,j})_{(i,j)\ \in A}$ vérifiant les deux conditions :

i) $0 \leq f_{i,j} \leq c_{i,j},\ \forall (i, j) \in A$;

ii) (conservation du flot)

$$\sum_{j:(i,j)\in A} f_{i,j} = \sum_{j:(j,i)\in A} f_{j,i},\ \forall i \in N \setminus (s, t)$$

On appelle *flot maximal* (ou *flot max*) un vecteur $(f_{i,j})_{(i,j)\in A}$ tel que pour tout autre flot $(f'_{i,j})_{(i,j)\in A}$, $f'_{i,j} \leq f_{i,j}$ pour tout $(i, j) \in A$.

Proposition 1 : Pour tout graphe G fini il existe un flot max.

Démonstration : On chercher à prouver L'existence d'un plus grand élément de $\mathcal{F} = \{(f_{i,j})_{(i,j)\in A} | f \text{est un flot } \}$.
D'abord, on voit que $\mathcal{F}$ n'est pas vide car le vecteur $(0)_{(i,j)\in A}$ est dans $\mathcal{F}$.
On remarque que $\mathcal{F}$ est fini car $\mathcal{F} \subset \times_{(i,j)\in A}\{0, ..., c_{i,j}\}$ et $c_{i,j} \in C$ est fini pour tout $(i, j) \in A$. Or tout ensemble fini admet un plus grand et un plus petit élément au sens de l'inclusion.

Proposition 2 : Pour tout flot $f\ \in \mathcal{F}$ dans G, on a l'égalité :

$$\sum_{j:(s,j)\ \in A} f_{s,j} = -\sum_{i:(i,t)\in A} f_{i,t}.$$

Démonstration : Par principe de conservation du flot $\sum_{j:(i,j)\ \in A} f_{i,j} = \sum_{j:(j,i)\in A} f_{j,i}$ pour $i \neq s$. Mais en particulier $f_{s,i} - \sum_{j:(i,j)\ \in A} f_{i,j} = 0$.

On appelle *valeur du flot* f, notée $v(f)$, l'élément :

$$v(f) = \sum_{(s,j)\ \in A} f_{s,j} = - \sum_{(i,t)\in A} f_{i,t}$$

Par le principe de conservation du flot cette valeur existe. Quand on cherche le flot max, on notera simplement v la valeur de flot recherchée.

2 Programme linéaire

2.1 Formulation

On souhaite calculer le flot maximal via la méthode linéaire en utilisant le solveur `glpsol`.

On cherche donc à maximiser v définit dans 1.2. Avec les notations de l'introduction soit $f_{i,j}$ la quantité de flot circulant sur l'arc $(i,j) \in A$, et $c_{i,j}$ la capacité maximale de cet arc.

Le problème de flot maximum peut être formulé comme suit :
Maximiser :

$$\sum_{(s,j)\in A} f_{s,j}$$

sous les contraintes :

$$\sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = \begin{cases} -v & \text{si } i = s \\ v & \text{si } i = t \\ 0 & \text{sinon} \end{cases}$$

$$f_{ij} \leq c_{ij}\ \forall (i,j) \in A$$

$$f_{ij} \geq 0\ \forall (i,j) \in A$$

Cette formulation caractérise entièrement notre problème car par la proposition 1, un flot max existe et s'il existe il doit alors vérifier la proposition 2 ; qui nous assure alors que c'est cet élément $\sum_{(s,j)\in A} f_{s,j}$ que l'on doit maximiser.
Les contraintes proviennent des conditions pour f d'être un flot.

2.2 Algorithme

On a considéré des *arcs jumeaux* comme étant les arcs ayant même noeud de départ et d'arrivée mais avec des capacités différentes.
Ainsi, on a notée le flot associé à ces arcs : $f\_i\_j\_c_{ij}$ pour chaque capacité. On ne néglige ainsi aucun arc dans notre résolution mais cela augmente le nombre de variables.

Structure du dossier :
Il y a un dossier `Linear` qui contient le fichier `generate_model.py`.
Les instances se trouvent dans `./src` et `generate_model.py` produit les modèles dans `./src`. On lance alors depuis le dossier `src` : `python3 Linear/generate_model.py inst-100-0.1.txt` Cela crée le fichier

`./model-100-0.1.lp`.
Afin de résoudre ceci on exécute la commande :
glpsol - -lp model-100-0.1.lp -o model-100-0.1.sol

3 Chemins augmentants

— une description de la méthode des chemins augmentant implémentée

Plusieurs algorithme on été choisit et testé.

Ford-Fulkerson :

Cette algorithme consiste à partir du flot nul $(0)_{(i,j)\in A}$ puis prendre un st-chemin, i.e. un chemin de s à t.
On sature le st-chemin avec le minimum de la capacité sur le chemin choisit.
Ensuite, on ...

:

:

4 Résultats

On a réalisé nos tests sur les systèmes d'exploitation : Windows 11 & MacOS Ventura.

Fichier	**Chemin augmentant (s)**	**Linear (s)**	**Écart (s)**
inst-2-0.25.txt	0.48001837730407715	-	-
inst-3-0.2.txt	0.44527459144592285	-	-
inst-3-0.22.txt	0.45975613594055176	-	-
inst-3-0.3.txt	0.4385108947753906	-	-
inst-4-0.25.txt	0.4311044216156006	-	-
inst-99-0.1.txt	2.452136754989624	-	-
inst-100-0.1.txt	2.3835582733154297	0.1880660057067871	2.1954922676086426
inst-101-0.1.txt	2.2259793281555176	-	-
inst-100-0.2.txt	5.815276145935059	0.3773059844970703	5.437970161437988
inst-200-0.1.txt	30.330101013183594	1.0871500968933105	29.242950916290283
inst-100-0.3.txt	13.9296293258667	0.6906599998474121	13.238969326019287
inst-300-0.1.txt	149.61344861984253	5.038102865219116	144.5753457546234
inst-200-0.2.txt	139.8529920578003	3.298656940460205	136.5543351173401
inst-400-0.1.txt	499.61123180389404	12.10365080833435	487.5075809955597
inst-500-0.1.txt	-	29.506887912750244	-
inst-500-0.2.txt	-	120.00071883201599	-
inst-500-0.2.txt	-	382.6500413417816	-

4.1 Critique des résultats

Déjà on constate qu'on n'arrive pas à dépasser les 500 nœuds en temps raisonnable. On voit également que le calcul par méthode linéaire est beaucoup plus rapide. Comme il s'agit simplement d'écriture de string dans un fichier texte cela paraît raisonnable. Néanmoins, nous sommes surpris que l'on ne puisses traiter que 500 nœuds avec une densité de 0.2 avec de simples écritures comme celles-ci.

Cette faible capacité de calcul peut être liée à une mauvaise optimisation d'écriture dans le cas linéaire. On n'a mis qu'une seule écriture dans le fichier `inst-xxx-0.i.lp` par le script de `model_generation.py` mais ceci ne suffit pas à l'optimiser.

identifiez si une m´ethode semble plus appropri´ee.

Un nombre exhaustif d'instances est fournies. Le rapport ne doit pas contenir les r´esultats de toutes les instances. S´electionnez les instances afin d'ˆetre le plus complet dans vos r´esultats.

4.2 Conclusion

On se rend compte sur un problème aussi limité et organisé que le problème du flot maximal à partir de 500 noeuds avec une densité de 0.2 nos machines n'y arrivent plus.

Il doit y avoir une possibilité d'optimisation. Mais est-ce que la meilleure optimisation nous permettrait de faire tourner toutes ces instances ? Comment le mesurer ?

Avec ce projet on se rend compte

Références

[1] bibtex

[2] L'écriture du code a été accélérée à l'aide du plugin Github Copilot

[3] Rédaction scientifique