université de BORDEAUX

Collège Sciences et Technologies
UF Mathématiques et Interactions

Réseaux de flots

Blayac Basile
Boyer Cyprien
Debouté Martin
Guimberteau Chloé
Pinet Pierre

CMI OPTIM

2021 – 2022

Algorithmes de flots

Table des matières

1 Introduction

En théorie des graphes, le problème de flot maximum consiste à trouver un flot réalisable d'un sommet source à un sommet puits tel que le flot soit maximal. Nous avons implémenté les algorithmes de plus court chemin augmentant (4.1.1 Shortest Augmenting Path Algorithm) et de poussage/réétiquetage (4.1.2 Preflow-Push Algorithm).

Le problème de coût minimum consiste à trouver un flot réalisable qui satisfait des contraintes de demande des sommets tout en minimisant le coût des arêtes empruntées. Nous avons implémenté l'algorithme Cycle cancelling (4.2.1).

Les algorithmes choisis sont présentés dans le livre *Network flows : theory, algorithms, and applications* de R.K. Ahuja et Orlin, 1993.

Nous avons écrit le programme de ce projet en C++. Pour commencer, nous avons généré des instances au format DIMACS et créé une structure de graphe, présentés ci-après (2). Nous avons ensuite implémenter les programmes linéaires (PL) de chaque problème (3) et pour finir nous avons comparé nos algorithmes aux PL et entre eux.

2 Données et mémoire

2.1 Gestion en mémoire des instances

On utilise des objets pour gérer nos instances dans le programme. Les graphes, sommets et arêtes ont chacun leur classe. Un graphe a en attribut un vecteur de sommet et un vecteur d'arêtes. Chaque sommet et chaque arête ont un identifiant sous forme d'un entier, cet identifiant correspond à leur position dans le vecteur du graphe auquel ils appartiennent.

Lorsque on veut faire référence à un sommet ou une arête, par exemple dans l'attribut $startId$ d'une arête, on utilise l'identifiant de l'objet auquel on veux faire référence.
Initialement on utilisait des pointeurs, cette pratique a été abandonnée car lorsque on ajoute un élément à un vecteur il change de place mémoire rendant ainsi invalide les pointeurs qui faisaient référence à des objets à l'intérieur du vecteur.

La relation entre l'identifiant et la position d'un objet rend le retrait d'un sommet ou d'une arête très compliqué, il faut changer les identifiants des objets qui le suivent dans le vecteur (leur position est diminuée de 1), il faut aussi changer toutes les références vers l'objet supprimé et les références vers les objets qui le suivent dans le vecteur.
On conserve cette relation malgré tout car dans le cadre de ce projet on ne change quasiment jamais les graphes après leur création.

Les arêtes ont un attribut $pairedEdgeId$ qui n'est utile que si elles appartiennent à un graphe résiduel. Cet attribut contient l'identifiant de l'arc résiduel inverse pour faire les changement de capacité résiduelle plus rapidement (l'usage d'un pointeur aurait été encore plus efficace).

2.2 Format des instances

2.2.1 Coût minimal

Les instances de coût minimum sont au format DIMACS (*DIMACS* s. d.).

Lignes de commentaires : les lignes de commentaires donnent des informations lisibles par l'homme sur le fichier et sont ignorées par les programmes. Les lignes de commentaire peuvent apparaître n'importe où dans le fichier.

Chaque ligne de commentaire commence par un caractère minuscule c :

c Voici un exemple de ligne de commentaire.

Ligne de problème : il existe une ligne de problème par fichier d'entrée. La ligne de problème apparaît avant toute ligne de descripteur de nœud ou d'arc. Pour les instances de réseau de flots à coût minimum, la ligne de problème a le format suivant :

p min NŒUDS ARCS

Le caractère minuscule p signifie qu'il s'agit d'une ligne de problème. L'indicateur "min" identifie le fichier comme contenant des informations de spécification pour un problème de flot à coût minimum. Le champ NODES contient une valeur entière spécifiant n, le nombre de nœuds dans le graphe. Le champ ARCS contient une valeur entière spécifiant m, le nombre d'arcs dans le graphe.

Descripteurs de nœuds : pour les instances de graphes à coût minimum, les lignes de descripteur de nœud décrivent les nœuds d'approvisionnement et de demande, mais pas les nœuds de transbordement. Autrement dit, seuls les nœuds avec des valeurs d'approvisionnement de nœud non nulles apparaissent. Il y a une ligne de descripteur de nœud pour chacun de ces nœuds, avec le format suivant :

n ID FLOT

Le caractère minuscule n signifie qu'il s'agit d'une ligne de description de nœud. Le champ ID donne un numéro d'identification de nœud, un entier compris entre 1 et n. Le champ FLOT donne la quantité d'approvisionnement au nœud ID.

Descripteurs d'arc : il existe une ligne de description d'arc pour chaque arc du réseau. Pour une instance de graphe à coût minimum, les lignes de description d'arc sont de la forme suivante :

a SRC DST LOW CAP COST

Le caractère minuscule a signifie qu'il s'agit d'une ligne de description d'arc. Pour un arc orienté (v,w) le champ SRC donne le numéro d'identification du sommet source v, et le champ DST donne le sommet de destination. Les numéros d'identification sont des entiers compris entre 1 et n. Le champ LOW spécifie la quantité minimale de flot qui peut être envoyée le long de l'arc (v,w), et le champ CAP donne la quantité maximale de flot qui peut être envoyée le long de l'arc (v,w) dans un flot réalisable. Le champ COST contient le coût unitaire du flot envoyé le long de l'arc (v,w).

2.2.2 Flot maximal

Les instances de flots maximum sont aussi au format DIMACS (*DIMACS* s. d.).

Le format est sensiblement le même à la différence que c'est un problème de maximisation (max) et qu'il n'y a que deux descripteurs de nœuds qui servent à indiquer la source (s) et le puits (t) comme ceci :

n ID s
n ID t

Les descripteurs d'arcs contiennent moins d'informations et sont de la forme :

a SRC DST CAP

2.3 Génération des instances

Les générateurs proviennent du projet Netflow (*Netflow* s. d.).

2.3.1 Flot maximal

Le générateur d'instances pour le problème de flot maximal provient du sous-projet/dossier waissi de Netflow. Le générateur est codé en Pascal et permet de créer un problème de flot maximum sur un graphe créé aléatoirement. Une fois le fichier compilé pour obtenir un exécutable nous avons écrit un script Shell (Bash) permettant de créer le nombre d'instance que nous voulons pour un certain type de graphe (*cf.* le fichier readme).

2.3.2 Coût minimal

Pour le problème de coût minimum des instances déjà existantes et provenant du sous-projet/dossier gte de Netflow furent utilisées (générateur gte) pour les tests. Mais nous avons également utilisé un convertisseur de problème de flot maximum vers coût minimum écrit en script Awk provenant du projet Netflow (cependant modifié par nos soins) afin de créer de nouvelles instances de problèmes de coût minimum à partir de nos instances de problème de flots maximum. De plus nous avons écrit un script en Python afin d'assigner des coûts aléatoire aux différents arcs ainsi qu'un script Shell (Bash) afin d'automatiser le tout. Dans cette méthode de génération les fichiers d'instances optimisent sur un flot de 1 mais nos algorithmes optimisent sur le flot maximal.

2.4 Format des solutions

La solution d'un problème de flot maximal et la solution d'un problème de coût minimal sont toutes les deux des affectations de flots à des arcs, on se sert donc d'un seul format de solution.
Si aucun flot ne passe sur un arc on considère qu'il n'y a pas d'affectation pour cet arc.

Sur la première ligne se trouve le temps en secondes qu'il a fallu pour trouver la solution, -1 si aucun temps n'a été donné lors de la sauvegarde.
Sur la deuxième ligne se trouve le nombre d'affectations de la solution.
Chaque ligne suivante est une affectation.
Sur une ligne d'affectation on trouve l'identifiant du sommet de départ de l'arc, l'identifiant du sommet d'arrivée de l'arc et la valeur du flot de sur l'arc.

Dans ce format de solution si il y a des arc parallèles on ne précise pas sur lequel se trouve le flot. Le choix de l'arc à remplir pour retrouver la solution est trivial, dans les problèmes de coût minimal il suffit de mettre le flot en priorité dans l'arc de plus bas coût.

3 Modélisation linéaire

On souhaite pouvoir vérifier la validité de nos algorithmes, pour cela on compare les valeurs objectifs avec celles de modèles linaires.
Pour la modélisation linéaire des problèmes de flot, on utilise le solveur Gurobi (API C++).

3.1 Flot maximal

3.1.1 Dimensions

$E = \{0, ..., n-1\}$ l'ensemble des sommets du graphe.
$V = \{0, ..., m-1\}$ l'ensemble des arêtes du graphe.

3.1.2 Données

n nombre de sommets.
m nombre d'arcs.
u_{ij} capacité maximale de l'arc (i, j).
s sommet source du flot.
t sommet puits du flot.

3.1.3 Variables de décision

f_{ij} le flot passant du sommet $i \in E$ au sommet $j \in E \backslash \{i\}$.
v le flot total.

3.1.4 Modèle

$$\max \quad v \tag{1}$$

$$sc \quad \sum_{j:(j,i)\in E} f_{ji} = \sum_{j:(i,j)\in E} f_{ij} \qquad \forall i \in V \tag{2}$$

$$\sum_{i:(s,i)\in E} f_{si} = v \tag{3}$$

$$\sum_{i:(i,t)\in E} f_{it} = -v \tag{4}$$

$$\sum f_{ij} \leqslant u_{ij} \qquad \forall (i,j) \in V \tag{5}$$

$$f_{ij} \in \mathbb{R} \qquad \forall (i,j) \in V \tag{6}$$

$$v \in \mathbb{R} \tag{7}$$

L'objectif (1) vise à maximiser le flot total v.
La contrainte (2) vérifie que les flots d'entrée et de sortie d'un sommet sont égaux.
Les contraintes (3, 4) définissent la variable de décision v.
La contrainte (5) vérifie que la capacité maximale de chaque arc est respectée.

3.2 Coût minimal

3.2.1 Dimensions

$E = \{0, ..., n-1\}$ l'ensemble des sommets du graphe.
$V = \{0, ..., m-1\}$ l'ensemble des arêtes du graphe.

3.2.2 Données

c_{ij} coût de l'arc entre les sommets i et j.
b_{ij} la demande ou la production de flot du sommet i.
u_{ij} capacité maximale de l'arc (i, j).
n nombre de sommets.
m nombre d'arcs.
s sommet source du flot.
t sommet puits du flot.

3.2.3 Variable de décision

f_{ij} le flot passant du sommet $E \in I$ au sommet $j \in E \backslash i$.

3.2.4 Modèle

$$\min \sum_{(i,j)\in E} f_{ij} * c_{ij} \tag{8}$$

$$sc \sum_{j:(i,j)\in E} f_{ij} - \sum_{j:(j,i)\in E} f_{ji} = b_i \qquad \forall i \in V \tag{9}$$

$$f_{ij} \leqslant u_{ij} \qquad \forall (i,j) \in E \tag{10}$$

$$f_{ij} \in \mathbb{R} \qquad \forall (i,j) \in V \tag{11}$$

$$\tag{12}$$

L'objectif (8) vise à minimiser la somme des coûts du flot.
La contrainte (9) vérifie que le flots d'entrée et de sortie d'un sommet respecte sa demande ou sa production.
La contrainte (10) vérifie que la capacité des arcs est bien respectée.

4 Algorithmes

4.1 Flot maximal

4.1.1 Shortest Augmenting Path Algorithm

Le Shortest Augmenting Path est un algorithme de flot max : il vise à assigner un flot à un graphe donné de sorte à ce que le flot arrivant sur le sommet puits soit maximum.
On cherche un chemin d'arêtes à capacités résiduelles strictement positives du sommet source au sommet puits, on augmente toutes les valeurs des capacités résiduelles des arêtes de la même valeur, jusqu'à ce qu'au moins une arête ait une capacité résiduelle nulle.
Pour cet algorithme, il faut faire attention à ne pas avoir de sommet avec des arêtes entrantes mais pas d'arêtes sortantes. On les supprime en amont de l'algorithme.

Complexité : $O(n^2m)$

4.1.2 FIFO Preflow-Push Algorithm

L'algorithme Preflow-Push consiste à "pousser" au maximum le flot à partir du sommet source jusqu'à atteindre le sommet puits. Pour obtenir un flot réalisable, on considère que le sommet source peut envoyer un nombre infini de flot et que le sommet puits peut en recevoir un nombre infini également.
On utilise l'attribut $exceedingFlow$ d'un sommet pour stocker le flot en excès e_s sur ce sommet s :

$$e_s = \sum flot\ entrant(s) - \sum flot\ sortant(s)$$

À chaque itération, on traite un sommet en excès, en commençant par le sommet source.

Complexité : $O(n^3)$

4.2 Coût minimal

4.2.1 Cycle cancelling

Le Cycle cancelling est un algorithme de coût minimum selon : un graphe comme celui d'un flot max avec en plus des coûts sur chaque arête et une valeur (qu'on nommera flot d'excès) pour chaque sommet correspondant au flot devant être créé/consommé par ce sommet (selon si la valeur est négative ou positive).

Pour réaliser cet algorithme sur une instance, on doit d'abord lui assigner un flot réalisable. Qui doit donc respecter les conditions classique d'un flot max mais qui en plus doit s'assurer que la somme de tous les flots entrants/sortants d'un sommet doit être égale à la valeur de son flot d'excès.

Pour ce faire, on choisit ici de créer deux sommets qui seront les source et puits du graphe, on les relie aux sommets avec des flots d'excès non nuls (> 0 pour source et < 0 pour puits) avec comme coût 0 et comme capacité le flot d'excès. On applique ensuite un flot max qui assignera les bons flots aux arêtes.

Il y a des subtilités supplémentaires comme le fait que pour notre flot max on ne doit pas avoir d'arêtes parallèles, or dans une instance de coût min deux arêtes peuvent avoir les mêmes sommets d'arriver et de départ avec des coûts différents (qui ne sont donc pas fusionnables). Il faut donc adapter le graphe avant de lui appliquer un flot max.

Une fois les flots assignés, il a été prouvé qu'une méthode pour trouver le flot de coût minimum est de "supprimer" tous les cycles du graphe qui ont un coût négatif. Le Cycle cancelling trouve un cycle négatif, le supprime et répète cette opération jusqu'à ce qu'il n'y est plus de cycle.

Pour trouver un cycle négatif, il faut chercher le chemin le plus court entre un sommet arbitrairement choisit et tous les autres sommets du graphe. (Il y a une subtilité ici car depuis un sommet, tous

les sommets ne sont pas obligatoirement accessibles : on ne peut donc pas définir toutes les distances. Il faut donc répéter ce processus avec un sommet dont la distance n'a pas été calculée avec tous les sommets accessibles depuis celui-ci, dont la distance n'a pas déjà été calculée).

On ne rentre pas dans les détails du calcul pour trouver les chemins les plus courts, mais si il y a un cycle négatif, certaines distances ne seront pas définies : le calcul ne cessera jamais en diminuant leur valeur en continue. Un sommet qui voit sa valeur diminuer plus de fois qu'il n'y a de sommets dans le graphe fait partie d'un cycle négatif, et on retrouve un cycle négatif dans la chaîne de ses prédécesseurs (instanciée pendant le calcul des distances).

Une fois un cycle négatif trouvé, on le supprime en modifiant le graphe résiduel jusqu'à ce qu'une de ses arêtes ait une capacité résiduelle nulle.

Complexité : $O(mCU)$

5 Expérimentations et résultats

Pour nos tests on utilise des échantillons de taille 10 pour chaque dimension de graphe que l'on souhaite tester. Cette taille réduite d'échantillon est due à la lenteur de la génération des instances et au fait que toutes les instances générées ne sont pas réalisables. Cependant le temps de résolution des algorithmes varie très peu d'une instance à une autre donc les résultats restent fiables.

Tous les tests ont été réalisés sur un processeur Processeur Intel Xeon W-1290.

5.1 Algorithmes de flot maximal

Les tests à l'heure actuelle on été réalisés sur l'algorithme de Shortest Augmenting Path, ces données pourront éventuellement être comparées avec des résultats sur Preflow Push une fois ce dernier algorithme débuggé/corrigé.

	4000 sommets	6000 sommets	8000 sommets	10000 sommets
arêtes = 2*sommets	5,5s	13,4s	23,6s	37,5s
arêtes = 10*sommets	97,4s	226,1s	382,8s	ND

Le tableau ci dessus donne le temps de résolution moyen en seconde de Shortest Augmenting Path.

5.2 Algorithmes de flot de coût minimal

Les tests à l'heure actuelle on été réalisés sur l'algorithme de Cycle cancelling, ces données pourront éventuellement être comparées avec des résultats sur Mean Cycle cancelling un fois ce dernier algorithme débuggé/corrigé.

	4000 sommets	6000 sommets	8000 sommets	10000 sommets
arêtes = 2*sommets	3,5s	6,8s	9,0s	28,5s
arêtes = 10*sommets	38,1s	71,7s	107,0s	220,9s

Le tableau ci dessus donne le temps de résolution moyen en seconde de Cycle cancelling.

Ces résultats sont troublant considérant le fait que Cycle cancelling doit d'abord faire une algorithme de flot maximal et que notre version utilise un Shortest Augmenting Path.

6 Conclusion

De nombreuses difficultés ont été rencontrées dans l'implémentation. Certains de ces problèmes proviennent des limitations des machines : lors du calcul de la fonction objectif sur certaines grosses instances on dépassait la taille maximale d'un entier, de plus dans Cycle cancelling l'espace mémoire occupé grandit considérablement avec le nombre d'arêtes.

La grande majorité des bugs rencontrés viennent de types de graphes spécifiques dont le traitement est implicite dans le livre sur lequel nous nous sommes appuyés.

Dans la partie résultat on ne compare pas les temps obtenus avec les performances de la résolution des PL avec un solveur, c'est parce que le solveur résout les modèles en moins d'une seconde même sur les instances de 10 000 sommet et 100 000 arêtes.

On s'attendait à ce que les algorithmes soient plus performant sur les grosses instances, et c'est probablement le cas si les algorithmes sont implémentés de la meilleure manière possible. Ce constat montre que nos implémentations sont largement améliorable.

Références

DIMACS (s. d.). `http://lpsolve.sourceforge.net/5.5/DIMACS_maxf.htm`.

Netflow (s. d.). `http://archive.dimacs.rutgers.edu/pub/netflow/`.

R.K. Ahuja, T.L. Magnanti et J.B. Orlin (1993). *Network flows : theory, algorithms, and applications*. Prentice Hall.