MO102 : Optimisation dans les réseaux de télécommunications

Mathieu Verchère
mathieu.verchere@ensta-paris.fr

Année scolaire 2021/2022

1 Modalités d'évaluation du cours

Le cours est organisé comme suit : une bonne partie des premières séances sera consacrée à de l'apprentissage pur et simple de Matlab à l'aide de feuilles d'exercice, puis nous passerons de plus en plus de temps sur le projet jusqu'à ne plus faire que du projet.

L'évaluation est en très grande partie au **contrôle continu** - ne faites pas tout au dernier moment ! Si jamais vous étiez absent.e lors d'une séance, prévenez-moi ou donnez-moi une courte explication, car votre présence a une vraie influence sur la régularité de votre avancée dans le projet et donc sur votre note finale.

Seuls 4 points sur 20 sont accordés lors de la dernière séance, dédiée aux **soutenances**. Il s'agit simplement de présenter votre travail sur le projet en dix à quinze minutes de présentation et cinq minutes de questions environ (le timing sera affiné en fonction du nombre de groupes à faire passer). Nous en reparlerons en séance, mais il s'agit simplement de présenter le sujet, ce que vous avez fait dessus, avec renfort d'extraits de code si nécessaire, ou de schémas/figures. Il est certes important de faire une présentation de bonne qualité, mais ne vous faites pas trop de soucis non plus, en général ces présentations se passent bien.

2 Introduction

Un réseau de télécommunication est un ensemble de noeuds (commutateurs, routeurs,....) qui sont reliés par des arcs (liaison de communication) de telle sorte que des messages puissent être transmis d'un bout à l'autre du réseau.
Il existe de nombreux problèmes réels qui nécessitent d'étudier un problème d'optimisation dans les réseaux. Dans ce projet, nous allons étudier deux problèmes très classiques: le problème du **plus court chemin** et celui du **flot maximum**.

Considérons, par exemple, le réseau de télécommmunication suivant:
À tout instant un message prend une certaine quantité de temps (indiquée par un nombre au dessus de chaque lien) pour traverser une ligne et se déplacer dans le réseau. Supposons qu'on veuille faire parvenir à t un message venant de s. On cherche alors la *meilleure route* dans le réseau qui prend le moins de temps pour acheminer le message. On cherche ainsi le plus court chemin de s à t dans le réseau.

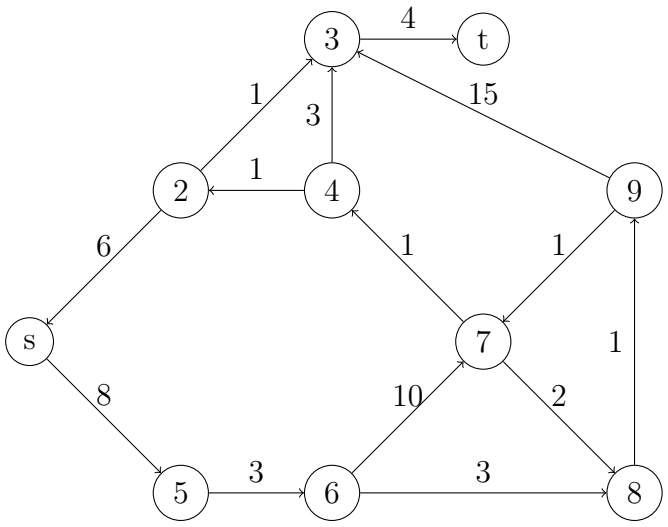

Figure 1: Exemple 1

Considérons maintenant le réseau ci-dessous où les nombres au dessus des arcs indiquent une capacité, c'est à dire la quantité maximum de donnée que l'on peut faire passer par ce lien. Le problème du flot maximum entre s et t consiste à trouver comment faire passer le plus de données possible entre s et t dans le réseau de façon à ce que pour chaque lien, la quantité de donnée sur le lien ne dépasse pas la capacité du lien.

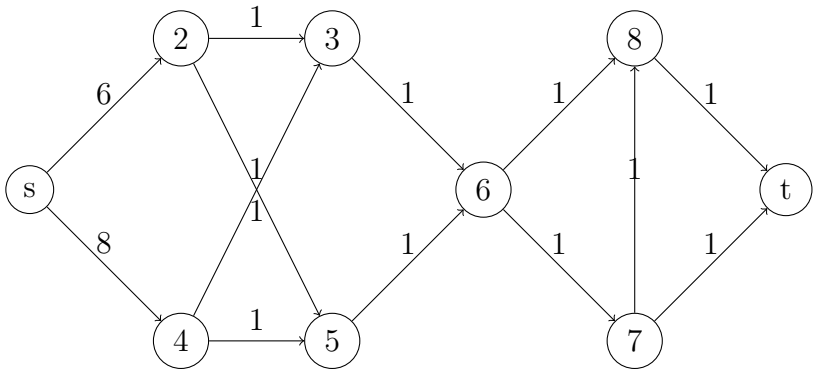

Figure 2: Exemple 2

Un peu de vocabulaire qui sera utilisé à travers le projet : on modélise le réseau par un **graphe orienté** $G = (V, E)$, où $V = \{1, ..., n\}$ est l'ensemble des **sommets** et $E \subseteq V^2$ est l'ensemble des **arcs** $(i, j) \in V^2$ (on dit alors que i et j sont **voisins** ou **adjacents**). Nous avons en outre, pour tout arête $(i, j) = e \in E$ du graphe, une valuation $c_e \in \mathbb{R}^*$ associée à e. Dans ce projet, cette valuation représentera soit une durée (ou plus généralement un **coût** ou **poids**) pour le problème du plus court chemin, soit une **capacité**, pour le problème du flot de valeur maximum.

3 Problème du plus court chemin

Dans cette partie, on s'intéresse au problème du plus court chemin dans un réseau: étant donnés deux sommets s et t du graphe, on cherche le plus court chemin (c-à-d la plus courte route) dans le graphe entre s et t, où nous appelons *longueur du chemin* la somme des valuations des arêtes du chemin.

Par exemple, dans le réseau de l'exemple 1, le plus court chemin entre s et t est $(s, 5, 6, 8, 9, 7, 4, 2, 3, t)$ et a pour valeur 23.

Nous allons d'abord voir deux algorithmes qui permettent de résoudre le problème, puis dans une seconde partie, nous allons considérer le cas ou il existe une contrainte de budget sur le choix du chemin.

Dans tout le sujet, un graphe sera représenté par sa matrice dite *d'adjacence valuée* A, où A est une matrice de taille $|V| \times |V|$ telle que $A_{ij} = c_{ij}$ si $(i, j) \in E$, $A_{ij} = 0$ sinon.

3.1 L'algorithme de Dijkstra

Nous allons, dans cette partie, implémenter l'algorithme de Dijkstra qui permet de calculer le plus court chemin d'un sommet s du graphe à tous les autres sommets du graphe. L'algorithme, dont le pseudo code est donné ci-dessous, fonctionne de la manière suivante: initialement tous les voisins, j de s ont une distance à s valant w_{sj}, les autres sont notés ayant une distance à s valant l'infini. On note alors le sommet s comme traité. Parmi les sommets non-traités, on considère celui (appelons le k) ayant la distance s la plus courte. On regarde alors, pour tous les voisins j de k, si il est possible de réduire la distance courante entre s et j, en passant par le sommet k. On marque alors k comme marqué.

Pour implémenter l'algorithme, nous allons considérer trois tableaux, D, L et P, de taille $|V| - 1$, où D enregistre la distance courante entre s et un sommet j, L est un tableau dont les éléments positifs correspondent aux sommets qu'il reste à traiter, et où P est un tableau qui garde en mémoire le *prédecesseur* de tout sommet j, c'est-à-dire le sommet qui précède j sur le chemin le plus court connu entre s et j.

Notons que cet algorithme fonctionne uniquement lorsque les valuations des arcs sont des nombres positifs. Dans le cas où certains arcs ont des coûts négatifs, la validité de la solution donnée par l'algorithme ne peut être assurée.

3.2 L'algorithme de Bellman-Ford

Contrairement à l'algorithme de Dijkstra, l'algorithme de Bellman-Ford fonctionne en présence de coûts négatifs. Il est également capable de détecter des cycles de valeur négative, auquel cas il n'existe pas de plus court chemin.

Quelques questions pour trouver le principe de l'algorithme :

Pour tout noeud $j \neq s$ et tout entier k, on note $c(j, k)$ la valeur du plus court chemin entre s et j contenant au plus k arcs.

1. **Initialisation :** Que vaut $c(j, 1)$ pour tout noeud j ?
2. **Principe algorithmique :** Montrez que l'équation de récurrence ci-dessous est verifiée:

$$c(j, k) = \min \left(c(j, k-1), \min_{(i,j) \in E} (c(i, k-1) + w_{ij}) \right)$$

3. En déduire l'algorithme.

Algorithm 1 Algorithme de Dijkstra

Entrée : La matrice d'adjacence valuée A du graphe, le sommet de départ s, le sommet d'arrivée t
D: vecteur de taille $|V|$ des distances d'un sommet i à s, initialisé à l'infini
$D[s] := 0$
P: vecteur de taille $|V|$ tel que $P[k]$ = le prédecesseur de k dans le chemin connu le plus court de s à k. Initialisé à 0.
Pour tout voisin j de s: $D[j] := c_{sj}$, $P[j] := s$
L: vecteur de taille $|V|$ permettant de déterminer le sommet à traiter
$L := D$
while t n'est pas marqué **do**
    Soit k l'indice de l'élément de L ayant la plus petite valeur strictement positive
    for j voisin de k **do**
        if $D[j] > D[k] + c_{kj}$ **then**
            $D[j] := D[k] + c_{kj}$
            $P[j] = k$
        end if
    end for
    $L[k] = 0$ (k marqué)
end while
Return $D[t]$, P

4. Implémentez l'algorithme de Bellman-Ford. Montrez que l'algorithme peut détecter des cycles négatifs.

Il existe plusieurs façons différentes d'implémenter cet algorithme. Nous n'allons pas nous attarder ici sur trouver la façon la plus performante. N'hésitez pas à me demander conseil pour faire fonctionner votre propre version.

3.3 Bonus : le plus court chemin avec contraintes de ressources

NB : Ne traitez cette section qu'après avoir terminé le reste du projet.

On suppose maintenant que chaque arc (i, j) du graphe comporte une seconde valuation, $Q_{ij} > 0$, correspondant à une quantité de ressource utilisée en empruntant l'arc (i, j). Le problème consiste maintenant à trouver le plus court chemin entre s et t telle que le coût total d'utilisation du chemin soit inférieure à une constante q donnée, représentant un budget maximal associé au transport de s à t.

Nous allons mettre au point un algorithme similaire à celui de Bellman-Ford. La subtilité supplémentaire est que pour deux chemins allant du noeud s au noeud j, il faut comparer à la fois la distance parcourue et la quantité de ressources utilisées. Un chemin en domine un autre si et seulement si il est plus court et utilise moins de ressources. Il faut donc garder en mémoire, pour chaque chemin, la longueur du chemin et la quantité de ressources utilisées. Pour tout $q > 0$, $j \in V$, on note $c(j, q)$ le coût du plus court chemin entre s et j parmi ceux utilisant une quantité de ressource totale ingérieure ou égale à q.

1. **Principe algorithmique :** Montrez que l'équation de récurrence suivante est satisfaite :

$$c(j, q) = \min_{\{(i,j)\in E \ | \ Q_{ij} \leq q\}} (c(i, q - Q_{ij}) + w_{ij})$$

2. Implémentez l'algorithme en prenant garde à la règle de domination des chemins.

Comme pour Bellman-Ford, il existe plusieurs façons différentes d'implémenter cet algorithme. N'hésitez pas à me demander conseil pour programmer votre version !

4 Représentation graphique

Dans cette section, vous trouverez un petit "mode d'emploi" sur les étapes à suivre pour obtenir un rendu graphique de qualité. Par ailleurs, les éléments évoqués ici sont le minimum nécessaire attendu dans le projet. Le rendu est à faire en priorité pour le problème du plus court chemin mais peut être adapté au problème du flot maximum (section suivante).

IMPORTANT : d'un point de vue pédagogique, il me semble important d'utiliser les capacités graphiques de Matlab. Il vous est libre de travailler sur le rendu graphique à tout moment de votre travail; cependant, si vous manquez de temps pour tout faire, je vous conseille de vous concentrer sur le rendu graphique avant de passer au problème du flot maximum.

4.1 Représenter un graphe

Matlab dispose par défaut de deux classes permettant de représenter des graphes mathématiquement et graphiquement : ce sont les objets Graph et Digraph, pour les graphes non orientés et orientés respectivement. Quelques fonctions existent directement dans Matlab pour travailler sur ces objets. Cependant, ces classes ne permettent qu'assez peu de liberté sur la représentation graphique. Je vous propose donc de faire vos propres graphiques, "à la main".

4.1.1 Sommets du graphe

Les coordonnées des sommets sont données dans les fichiers exemples, mais vous pouvez toujours les changer si vous souhaitez modifier la disposition du graphe.

Je suggère de représenter les sommets par des cercles - les fichiers exemples contiennent d'ailleurs également les rayons de ces cercles. Cependant, si vous souhaitez expérimenter avec d'autres formes, libre à vous ! Je vous suggère également de créer une fonction qui s'occupe de tracer les traits adéquats pour un sommet, afin de la réutiliser dans le reste de votre code.

Enfin, pour s'y retrouver, il faut numéroter les sommets. Pour insérer du texte dans un graphique, je vous conseille de vous tourner vers la fonction 'text' : la commande text()x,y,str) permet d'afficher la chaîne de caractères str aux coordonnées (x, y).

4.1.2 Arêtes du graphe

Il s'agit ici de tracer une flèche entre deux points. Il nous faut bien une flèche et pas un simple trait car nous travaillons avec des graphes orientés. Il y a probablement plusieurs méthodes pour obtenir un bon résultat, mais voici deux conseils que toute bonne solution doit prendre en compte :

- Comme précédemment, isolez le tracé d'une arête dans une fonction à part. Vous pourrez la réutiliser dans d'autres parties de votre code pour le rendre plus lisible et éliminer des répétitions.

- Prenez en compte la géométrie que vous avez adoptée pour représenter les sommets afin de déterminer précisément les deux extrémités de votre flèche.

De même, vous voudrez probablement écrire quelque chose à proximité de votre arête, comme sa longueur dans le cas du plus court chemin par exemple. Je vous renvoie à nouveau vers la fonction 'text'.

4.1.3 Mise en évidence

Pour représenter la solution d'un problème, que ce soit le plus court chemin ou le flot maximal, vous aurez à mettre en évidence certains éléments : le départ / l'arrivée du chemin, le chemin emprunté...

Pour ceci, je vous conseille de mettre en évidence ces éléments en en modifiant la couleur, où éventuellement le style du trait. Je recommande donc que vos deux fonctions de tracé de sommet et d'arête prennent en argument la couleur de la même façon que le fait la commande 'plot' de Matlab.

5 Problème du flot maximum entre s et t

On cherche à présent à résoudre le problème du flot maximum entre deux noeuds dans un réseau. On rappelle que le flot correspond à une *circulation* dans le réseau entre l'entrée s et la sortie t de telle sorte que la quantité de flot circulant sur chaque arc ne dépasse pas la capacité de cet arc. La capacité d'un arc (i, j) est notée c_{ij}, et le flot passant par cet arc est noté f_{ij}. Dans l'Exemple 2, la valeur d'un flot maximum est de 2.

Pour résoudre ce problème, nous allons implémenter **l'algorithme de Ford-Fulkerson** dont le principe est le suivant:

- On part d'un flot **réalisable**, respectant les contraintes de capacité de chaque arc (par exemple, le flot nul).
- On utilise la procédure de marquage suivante:
  - le sommet s est marqué
  - Soit i marqué et soit $(i, j) \in E$ tel que la capacité sur (i, j) soit strictement supérieure à la valeur actuelle du flot sur (i, j) et j non marqué. Alors j est marqué $+$.
  - De même, soit $(j, i) \in E$ tel que le flot sur (j, i) soit strictement positif et j non marqué. Alors j est marqué **-**.
  - On garde en mémoire le prédecesseur de chaque sommet, c'est-à-dire que si i est marqué et permet de marquer j, i est le prédecesseur de j.
- Si le noeud t est marqué, alors on a trouvé une **chaîne augmentante** allant de s à t. Pour trouver la composition de la chaîne on utilise les prédecesseurs. Cette chaîne est susceptible de traverser certains arcs à l'envers étant donné la règle de marquage. On peut alors modifier la valeur du flot le long de la chaîne :
  - Soit M^+ l'ensemble des arcs traversés **dans le bon sens** par la chaîne augmentante. Soit M^- l'ensemble des arcs traversés **dans le sens inverse** par la chaîne augmentante.

- Alors pour tout arc de M^+, il faut **augmenter** le flot de $f_{mod} = min(\min_{(i,j)\in M^+} c_{ij} - f_{ij}, \min_{(i,j)\in M^-} f_{ij})$. Pour tout arc de M^-, il faut **réduire** le flot de cette même quantité f_{mod}.

- On répète la procédure tant qu'il est possible d'exhiber une chaîne augmentante. Quand ce n'est plus possible, on a construit un flot maximal.

Implémentez l'algorithme. Comme toujours, l'implémentation peut prendre différentes formes.