AG41

PROJET D'OPTIMISATION EXACTE
Problème de transbordement

Esia Belbachir, Pierre Brunet de Monthélie

P2016

Table des matières

Table des figures

1 Analyse mathématique

1.1 Paramètres

V Ensemble des noeuds du graphe tel que $V = C \cup F \cup P$

C Ensemble des noeuds clients tel que $\forall i \in C, b_i > 0$

P Ensemble des noeuds plateformes tel que $\forall i \in P, b_i = 0$

F Ensemble des noeuds fournisseurs tel que $\forall i \in F, b_i < 0$

A Ensemble des arcs orientés $e = (i,j) \in V^2$ tel que tous les fournisseurs ont un arcs vers toutes les plateformes et toutes les plateformes ont un arc vers tous les clients :
$\forall (i,j) \in A,$ alors $(i \in C$ et $j \in P)$ ou $(i \in P$ et $j \in C)$

b_i Quantité de produits demandée par le noeud $i \in V$
$\sum_{i \in V} b_i = 0$

g_i Coût de transbordement unitaire de la plateforme $i \in P$

s_i Temps de transbordement de la plateforme $i \in P$

u_{ij} Capacité de l'arc $(i,j) \in A$

c_{ij} Coût fixe d'utilisation de l'arc $(i,j) \in A$ si cet arc transporte des produits

h_{ij} Coût unitaire de transport de l'arc $(i,j) \in A$

t_{ij} Temps de transport de l'arc $(i,j) \in A$

T Délai maximum pour acheminer tous les produits

1.1.1 Remarques

- $C \cap F \cap C = \varnothing$
- $\forall i \in V, b_i$ est égale à la différence entre la quantité de produits qui doit rentrer et la quantité de produits qui doit sortir du noeud i

1.2 Variables

x_{ij} Quantité de produits transportés sur l'arc $(i,j) \in A$

x_{ijk} Quantité de produits transportés sur l'arc (i,j) puis sur l'arc (j,k) avec $i \in F, j \in P$ et $k \in C$
(i,j,k) représente le chemin emprunté par un paquet de x produits

y_{ij} Valeur binaire égale à 1 si l'arc $(i,j) \in A$ est utilisé pour transporter des produits, 0 sinon

y_{ijk} Valeur binaire égale à 1 si le chemin $(i,j,k) \in F \times P \times C$ est utilisé pour transporter un paquet de produits, 0 sinon
Si (i,j) et (j,k) sont utilisés, mais pas pour le même paquet, y_{ijk} est égal à 0

1.3 Fonction objectif

$$\min z = \sum_{(i,j) \in F \times P} y_{ij}(c_{ij} + x_{ij}(h_{ij} + g_i)) + \sum_{(i,j) \in P \times C} y_{ij}(x_{ij}h_{ij} + c_{ij})$$

1.4 Contraintes

$$
\begin{array}{rl}
\text{Temps} & \forall (i,j,k) \in (F \times P \times C), y_{ijk}(t_{ij} + t_{jk} + s_i) \leqslant T \\
\text{Capacité} & \forall (i,j) \in A, x_{ij} \leqslant u_{ij} \\
\text{Conservation} & \forall j \in V, \sum\limits_{\substack{k \in V \\ (j,k) \in A}} x_{jk} + \sum\limits_{\substack{i \in V \\ (i,j) \in A}} x_{ij} = b_i \\
\text{Relation variables} & \forall (i,j) \in F \times P, x_{ij} = \sum\limits_{k \in C} x_{ijk} \\
& \forall (j,k) \in F \times P, x_{jk} = \sum\limits_{i \in F} x_{ijk} \\
& \forall (i,j) \in A \text{ si } x_{ij} > 0, \text{ alors } y_{ij} = 1 \\
& \forall (i,j,k) \in F \times P \times C, \text{ si } x_{ijk} > 0, \text{ alors } y_{ijk} = y_{ij} = y_{jk} = 1
\end{array}
$$

1.5 Représentation graphique

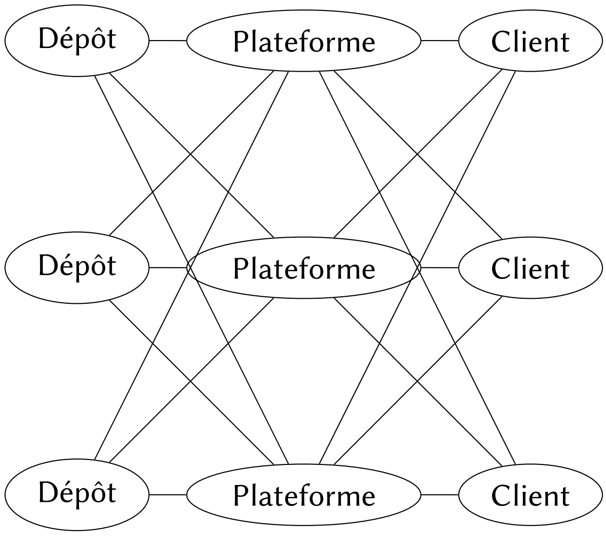

Figure 1 – Représentation du problème

2 Questionnement et algorithmes

En mettant en évidence les contraintes et les variables, nous avons identifié des caractéristiques permettant de fractionner et de simplifier le problème considéré.

2.1 Un problème de temps

Dans notre analyse mathématique, nous avons inclu la contrainte temporelle de la manière suivant : tout paquet de taille y passant par les noeuds i,j et k doit être acheminé dans un temps inférieur à une limite T. Afin de remplir cette contrainte, nous avons adapté la représentation graphique de ce problème de la manière suivante : chaque noeud plateforme est remplacé par un graphe biparti entre les arcs entrants et les arcs sortants.

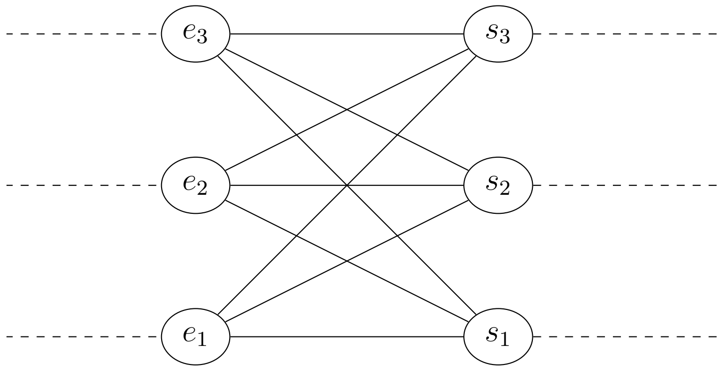

Figure 2 – Décomposition d'une plateforme avec 3 dépôts et 3 clients

Pour chaque noeud plateforme i :

- Caractéristique des noeuds créés j :
  - $b_j = 0$
- Caractéristiques des arcs créés (j, k) :
  - $t_{jk} = s_i$
  - $h_{jk} = g_i$
  - $c_{jk} = 0$
  - Si le chemin d'un fournisseur à un client enpruntant l'arc (j, k) a un temps total de transport supérieur à T, $u_{jk} = 0$
    Sinon, $u_{jk} = +\infty$

En représentant le problème ainsi, on peut "ignorer" la contrainte de temps puisqu'elle est intégrée dans la structure même du graphe.

2.2 Un problème de flot maximal à coût minimum

Une fois la contrainte de temps mise de côté, on constate que le problème proposé est en fait un problème de flux maximal à coût minimum.

2.2.1 Flot maximal

En ajoutant un noeud source et un noeud cible au graphe, on se retrouve face à un problème de flux maximal classique.

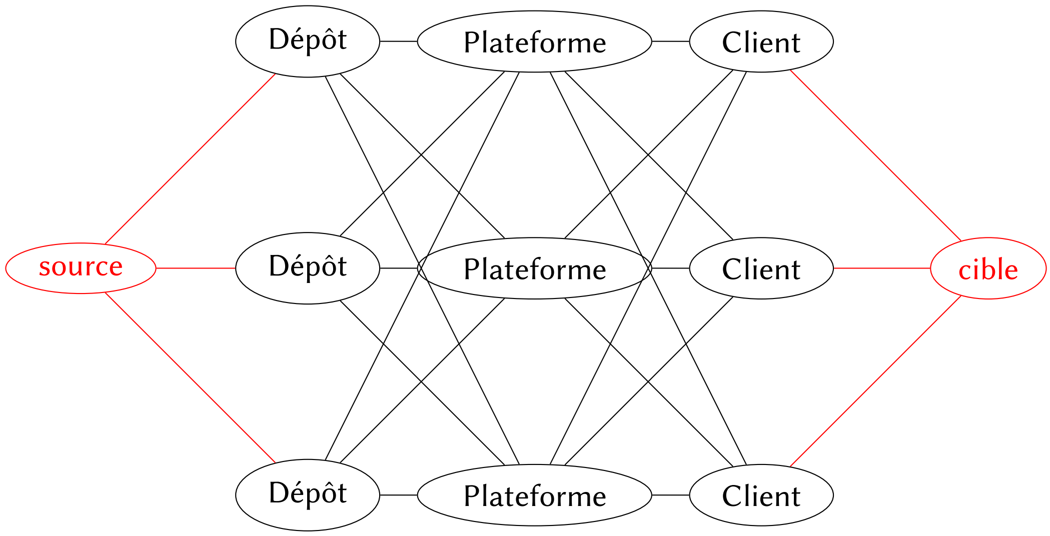

FIGURE 3 – Ajout de la source et de la cible sur le graphe pour faciliter le traitement

Pour résoudre ce problème, nous pouvons utiliser l'algorithme de Edmonds-Karp, implémentation de la méthode de Ford-Fulkerson. Il repose sur la recherche d'un chemin d'amélioration dans le graph résiduel.

Soient $c(u, v)$ la capacité de l'arc (u, v) et $f(u, v)$ le flot de (u, v).

```
dfs(G, u)
DEBUT
    POUR u dans V
        Marquer u
        POUR tous les noeuds v dans noeudsAdjacents(u)
            SI v n'est pas marqué
                dfs(G, v)
            FIN SI
        FIN POUR
    FIN POUR
FIN

fordFulkerson(G)
DEBUT
    POUR (u,v) dans Arcs(G)
        f(u,v) = 0
        Marquer s
        TANT QUE il existe une chaîne améliorante μ
            fluxDir = min(c(u,v)-f(u,v)),
                        où u est un arc direct de μ
            fluxInd = min(f(u,v)),
                        où u est un arc indirect de μ
            flux = min(fluxDir, fluxInd)
            POUR (u,v) dans ArcsDirects(μ)
                f(u,v) = f(u,v) + flux
            FIN POUR
            POUR (u,v) dans ArcsIndirects(μ)
                f(u,v) = f(u,v) - flux
```

```
            FIN POUR
        FIN TANT QUE
    FIN POUR
FIN
```

La complexité est égale à $m^2 * n$ avec m le nombre d'arcs et n le nombre de noeuds.

2.2.2 Coût minimum

Le problème de flux à coût minimum présenté ici est plus complexe que les problèmes de flux à coût minimum classiques car les arcs possèdent dans notre cas un coût unitaire et un coût fixe.

Nous avons alors décidé de calculer le coût unitaire à partir de la moyenne des coûts unitaire et fixe selon la capacité de l'arc.

Par exemple, pour une capacité de n produits, on aurait :

$$\frac{c_{moyen} = n * c_{unitaire} + c_{fixe}}{n}$$

Une solution usuelle d'un problème de flux à coût minimum est de chercher un cycle négatif dans le graphe résiduel des coûts. En utilisant les coûts moyens comme coûts unitaires, on peut utiliser cette méthode.

Une autre possibilité serait de résoudre le problème avec, tout d'abord, les coûts unitaires. On aurait alors une solution basé sur la quantité de produits. Ensuite, on améliorerait cette solution avec les coûts fixes.

DEBUT
    Initialement, $\phi = (0, ..., 0); G^e_\phi = R$
    FAIRE
        trouver C un chemin de coût minimal de s à p dans le graphe d'
        $\delta = min(r_{ij})(avecr_{ij} = c_{ij} - \phi_{ij})$
            $(i, j) \in C$
        POUR tout arc u = (i, j) de C FAIRE
            SI (i, j) arc du graphe
                FAIRE $\phi(u) = \phi(u) + \delta$
            SINON
                SI (j, i) arc du graphe, $\phi(u) = \phi(u) - \delta$
                    modifier le graphe d'écart G^e_ϕ
                FIN SI
            FIN SINON
            FIN SI
        FIN POUR
    TANT QUE il existe un chemin de s à p dans G^e_ϕ
FIN

3 Résumé

Afin de résoudre le problème proposé, nous avons segmenté sa résolution en 3 sous-problèmes : temps, flot maximal et coût minimum.

Les 2 premiers sont nécessaires à l'obtention d'une solution qui remplit les contraintes, le dernier permet d'améliorer la solution générée.

Pour trouver une solution, le temps minimum de calcul est donc celui de la construction du graphe et de la résolution du problème de flot maximal selon les méthodes expliquées plus haut.

Ensuite, plus on laissera le programme s'exécuter longtemps, plus la solution sera bonne, jusqu'à atteindre la solution optimale.

La complexité de ce programme est égale à la somme des complexités des différents algorithmes utilisés.

4 Implémentation

Pour pouvoir implémenter notre algorithme, nous avons décidé de créer notre propre structure de graphe afin qu'elle s'adapte aux différentes méthodes que nous avons utilisées. La séparation des plateformes a ainsi été directement prise en compte dans la structure.

Le programme est codé en Java.

4.1 La classe Graph

4.1.1 Graph : une classe générique

La classe Graph est une classe générique, ce qui permet une certaine flexibilité quant aux données portées par les noeuds et les arcs. Elle permet de stocker la structure du graphe indépendament des données. Le Graph fonctionne avec des numéros uniques identifiant les noeuds. Les arcs sont également identifiés en utilisant un couple d'identifiants.

Dans notre solver, nous n'avons utilisé que les classes Node et Edge en paramètre de classe de Graph, mais on pourait tout à fait utiliser d'autres couples <node type, edge type> pour améliorer les algorithmes.

4.1.2 Les classes Node et les Edge

Les classes Node et Edge sont composées des données du problème en terme d'informations portées par les noeuds et les arcs.

La classe Node se compose donc de ces données de base, mais aussi d'un attribut parent qui est utilisé dans le DFS permettant de générer tous les cycles du graphes. Il est utilisé pour remonter jusqu'au parent créant le cycle.

4.2 La lecture du fichier de problème

La lecture du fichier de problème est la 1er étape de l'algorithme. Elle permet de construire le graphe de base du problème. Elle permet aussi d'éliminer les arcs dont la capacité est nulle et de déterminer les 3 ensembles de noeuds que sont les fournisseurs, les plateformes et les clients. C'est également à cette étape qu'est détecté si le nombre de produits offerts par les founisseurs correspond au nombre de produits demandé par les clients.

4.3 La séparation des plateformes

La séparation des plateformes permet de prendre en compte la contrainte de temps. Pour la mettre en place en Java, on utilise la liste de noeuds plateformes crée à la lecture du fichier de problème.

Pour chaque plateforme, on crée de nouveaux noeuds comme nous l'avions prévu dans la phase d'analyse du problème de transbordement. Les noeuds initiaux sont supprimés à l'issu de cette étape, et 2 listes sont crées : la liste des plateformes gauches (liées aux fournisseurs) et la liste des plateformes droites (liées aux clients).

Une fois le graphe modifié, on vérifie que tous les chemins respectent la contrainte de temps total de transport. Si ce n'est pas le cas, l'arc entre la plateforme gauche et la plateforme droite inclue dans le chemin est supprimé.

Puisqu'on supprime les plateformes d'origine, il est impossible de savoir à quel plateforme d'origine correspond une plateforme gauche ou une plateforme droite. Il est donc nécessaire de créer une table associative sous la forme d'une HashMap qui à chaque nouvelle plateforme associe l'ancienne plateforme.

4.4 La création d'une solution initiale

La génération d'une solution initial s'effectue en 2 étapes.

D'abord le remplissage arbitraires d'arcs jusqu'à leur maximum. Pour cela, tous les chemins fournisseur-plateforme gauche-plateforme droite-client sont remplis au maximum jusqu'à ce que toute la demande soit remplie, ou bien jusqu'à ce qu'on arrive à la fin des chemins.

Si toute la demande n'a pas pu être totalement remplie, on ajoute une source et une cible au graphe et un DFS est lancé sur le graphe résiduel pour trouver des chemins d'amélioration. Comme la taille du problème est variable, on ne recherche pas le meilleur chemin d'amélioration, mais le 1er que l'on trouve, afin d'éviter de tout parcourir. Pour la même raison, nous avons décidé d'implémenter un DFS procédural et non récursif pour éviter un overflow de la pile.

Si on n'arrive pas à un flot permettant de répondre à toute la demande, alors le problème est insoluble.

4.5 Un diagramme de flot maximal à coût maximal

Pour cet algorithme, nous n'avons pas besoin de source et de cible car l'algorithme circule à partir de l'ensemble des dépôts. De plus, la source et la cible sont désormais inutile car, leur capacité étant égale au dépôt ou au client qui leur est associé, ils sont déjà chargé au maximum. On enlève donc ces deux noeuds à la fin de l'algorithme précédent.

À partir de cette solution, on va chercher à l'améliorer en cherchant les cycles négatifs comme nous avions décidé de le faire dans la première partie. En revanche, l'ajout unitaire de produits, pour chaque cycle, risque d'être très lent car certains arcs ont une capicité très importante avec une demande aussi importante.

Nous avons donc décidé de récupérer la capacité maximale du cycle et de calculer le graphe résiduel correspondant. À chaque itération, l'algorithme vérifie qu'il y a des cycles négatifs et que le temps maximal de calcul n'est pas atteint.

Lorsqu'un coût est négatif, le cycle est considéré comme négatif. On ajoute et soustraie alors les différents changements de flot de produits. Puis on recommence une itération si il reste du temps.

4.5.1 La recherche de cycles

Pour trouver tous les cycles d'un graphe, plusieurs méthodes sont possibles. Nous avons décidé d'utiliser le DFS car c'est la méthode nous semblant la plus logique et la plus concrète.

On lance un DFS récursif, on crée une liste de noeuds candidats et une liste de noeuds découverts. Si un noeud est trouvé une deuxième fois et s'il fait partie de la liste des candidats, on remonte jusqu'à ce qu'on trouve un noeud parent égale à ce même noeud.

Ensuite, on enlève tous les cycles qui sont trouvés plusieurs fois à cause du décalage. Enfin, on inverse l'ordre de chaque cycle.

Cette implémentation étant en récursif, elle est très puissante mais très gourmande en ressource. Ainsi, pour les graphes de 20 noeuds et plus, le DFS met énormément de temps à se faire. Le temps d'une itération est donc grandement augmentée.

4.5.2 La prise en compte des coûts fixes

Le principe du coût fixe est l'ajout d'un coût supplémentaire d'un arc dès que ce dernier est utilisé, même pour un produit. Notre but est donc de minimiser le nombre d'arcs utilisés afin d'éviter des coûts trop importants.

Pour prendre compte les coût fixes, il suffit d'ajouter, dans le calcul du coût d'un cycle, une condition afin d'ajouter ou de soustraire au coût total le prix fixe de l'arc.

4.6 Le temps de calcul

Comme on ne peut pas laisser tourner l'algorithme indéfiniement, un temps maximum de calcul est donné en paramètre au programme. De plus, pour plus de précision, nous avons indiqué le temps que chaque étape prend pour s'effectuer : la lecture de fichier, la création d'une solution initial (qui prend en compte la séparation des plateformes), l'amélioration de la solution.

Quel que soit le temps imparti, les 2 premières étapes sont toujours remplies afin que le programme retourne toujours au moins une solution. Si il reste du temps, on améliore la solution initiale. Comme pour la lecture de fichier et l'initialisation, une itération d'amélioration ne peut pas être interrompue. Ainsi, le temps total n'est pas forcément égal au temps de calcul imparti.

5 Résultats

Après avoir mené les tests dont les résultats sont indiqués ci-dessous, nous nous sommes rendus compte que les algorithmes de solution initial et d'amélioration étaient très longs. Le dédoublement des plateformes augmente en effet énormément la taille du graphe et les DFS sont donc très long. Changer d'algorithme ou trouver un moyen de ne pas dédoubler les plateformes seraient des moyens de palier à ce problème.

Fichier	Résultat	Temps
transshipment1.txt	212.0	5.397s
transshipment2.txt	pb dans la lecture du fichier	
tshp006-01.txt	pb dans la lecture du fichier	
tshp10-01.txt	3940.5911509852026	0.02s
tshp10-02.txt	pb dans la lecture du fichier	
tshp010-01.txt	4756.506448083509	157.089s
tshp010-02.txt	2126.9661327456147	1.856s
tshp010-03.txt	1951.8313355932821	0.079s
tshp010-04.txt	2114.3288184684357	0.888s
tshp010-05.txt	2866.751823637155	1.253s
tshp020-01.txt	4573.544241371602	stoppé au bout de 20 min
tshp020-02.txt	7128.861258575862	stoppé au bout de 35 min
tshp020-03.txt	9662.086844423093	stoppé au bout de 60 min
tshp020-04.txt	11159.946168444236	arrêté au bout de 20 min
tshp020-05.txt		
tshp050-01.txt		
tshp050-02.txt		
tshp050-03.txt		
tshp050-04.txt		
tshp050-05.txt		
tshp100-01.txt		
tshp100-02.txt		
tshp100-03.txt		
tshp100-04.txt		
tshp100-05.txt		
tshp500-01.txt		
tshp500-02.txt		
tshp500-03.txt		
tshp500-04.txt		
tshp500-05.txt		