Une heuristique pour résoudre des problèmes de flots insécables de grande taille

François Lamothe[1], Alain Hait[1], Emmanuel Rachelson[1] , Cedric Baudoin[2], Mathieu Gineste[2]

[1] ISAE-Supaéro, DISC, France
{francois.lamothe,emmanuel.rachelson,alain.hait}@isae-supaero.fr
[2] Thales Alenia Space, France
{cedric.baudoin,mathieu.gineste}@thalesaleniaspace.com

Mots-clés : *Flots insécables, heuristique*

1 Introduction

Le problème de flot insécable est une variante NP-complete du problème de flot maximum très étudiée dans la littérature. Les données de ce problème sont : un graph $G = (V, E)$ et un ensemble de commodités $(o_k, d_k, D_k)_{k \in K}$. Chaque commodité veut transmettre une demande D_k depuis une origine o_k jusqu'à une destination d_k à travers un chemin unique. Ce choix de chemin doit assurer que la capacité des arcs c_e n'est pas dépassée (ou que ce dépassement est minimisé).

Ce problème a été présenté par Kleiberg (1996) [1] pendant sa thése. Le problème de flot insécable est NP-complet car il contient les problèmes de sac à dos et de chemins à arcs disjoints comme cas particuliers.

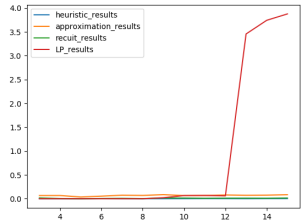
Ce problème possède de nombreuses applications, en particulier dans le domaine des télécommunications et du transport de marchandise. Dans ces applications, les instances à résoudre peuvent être très grande avec plus de 500 noeuds, 2000 arcs et 150 000 commodités. Dans la littérature, peu d'algorithmes sont capables de s'attaquer à des instances aussi grande. L'algorithme proposé par Raghavan et Thompson [2] en fait partie. Malheureusement, même si son facteur d'approximation, $O(\frac{log\ n}{log\ log\ n})$, est optimal, en pratique les résultats sont de mauvaise qualité.

C'est pourquoi nous nous sommes intéressés à créer un algorithme capable de résoudre de grandes instances en donnant de bons résultats en pratique. Notre algorithme est une heuristique et une extension de l'algorithm de Raghavan et Thompson [2]. Nous montrons expérimentalement que notre algorithme est capable de résoudre de grandes instances et que même s'il n'a pas de garanties théoriques sur la qualité de la solution fournie, en pratique, les résultats sont meilleurs que ceux de Raghavan et Thompson.

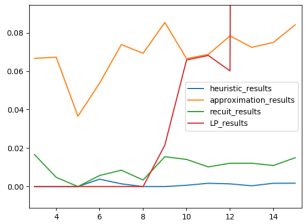
2 L'algorithme de Raghavan et Thompson

Leur algorithme possède deux étapes. Premièrement, la relaxation linéaire du problème est résolu : il s'agit du problème de flot multi-commodités. Ensuite pour chaque commodité, un seul chemin est choisi parmi ceux utilisés dans la relaxation linéaire. Pour cela, parmi l'ensemble P_k des chemins utilisés dans la relaxation linéaire, le chemin p_{ki} est choisi avec une probabilité $\frac{f_{ki}}{D_k}$ où f_{ki} est le flot mis par la commodité k sur le chemin p_{ki} et D_k est la demande de la commodité k.

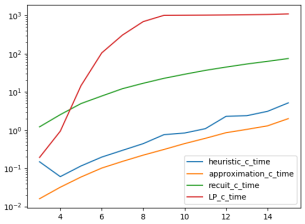
Cette procédure d'arrondi aléatoire renvoie avec une haute probabilité une solution insécable de congestion $O(log\ m/log\ log\ m)$ fois plus grande que la congestion de la solution fractionnelle. La congestion est le plus grand dépassement de capacité sur les arcs.



(a) Performance divisée par la demande totale



(b) Performance divisée par la demande totale : zoomé



(c) Temps de calcul en secondes

FIG. 1 – Performance et temps de calcul en fonction du paramètres N contrôlant la taille des instances : la capacité des arcs est de 8, la demande maximale d'une commodité est de 3.

3 Notre heuristique

Notre algorithme possède le même principe que l'algorithme de Raghavan et Thompson : résoudre la relaxation linéaire, choisir un seul chemins par commodité avec un procédure d'arrondi aléatoire de la solution fractionnelle. Cependant, il posséde deux différences. Premièrement, nous fixons le chemin des commodités par ordre décroissant des demandes des commodités. Cela permet de laisser les plus petites à la fin pour remplir les trous restants. Deuxièmement, nous actualisons la solution fractionnelle plusieurs fois au cours de l'algorithme après avoir fixé une partie des commodités.

L'actualisation est très importante. Elle permet à l'algorithme de choisir les chemins des commodités avec une information plus pertinente car actualisée. Nous avons décidé d'actualiser la relaxation dans les deux cas suivants :

— Depuis la dernière actualisation, le chemin de γ commodités a été fixé alors que la commodité utilisée plusieurs chemins dans la relaxation linéaire.

— Il n'y a plus de capacité restante sur les chemins disponibles pour la commodité considérée.

4 Résultats expérimentaux

Afin d'illustrer la performance de notre algorithme, nous l'avons comparé expérimentalement avec 3 algorithmes. Le premier est un modèle arc-noeud de PLNE résolu avec Gurobi 8.11. Le deuxième est un recuit simulé. A chaque itération, une commodité change de chemin et la solution est évaluée pour voir si elle s'est améliorée. Le troisième est l'algorithme d'approximation de Raghavan et Thompson.

Les instances ont été créées comme suit. Le graphe est une grille torique $N \times N$ avec N noeuds additionnels étant chacun connecté à $2N$ noeuds de la grille. Les noeuds additionnels servent d'origine aux commodités. Cette forme de graphe est induite par une application en télécommunication. Chaque arc a une capacité de 8 et les demandes des commodités sont de 1, 2 ou 3.

Références

[1] Jon M Kleinberg. Single-source unsplittable flow. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 68–77. IEEE, 1996.

[2] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms : approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2) :130–143, 1988.