AG41 - Problème de Transbordement

Guillaume PROST et Aurélien DURANCE

28 Mai 2016

1 Analyse Du Problème

Paramètres:
F: nombre de fournisseurs
D: nombre de destinataires
P: nombre de plateformes
E_{ij}: Edge allant de i vers j
g_i: coût d'utilisation de la plateforme i
c_{ij}: coût dfe transfert du node i vers le node j
h_{ij}: coût unitaire du trajet du node i vers le node j par objet
t_{ij}: temps de trajet du node i vers le node k en passant par le node j
s_i: temps de transbordement
Cap_D: capacité de reception d'un destinataire D
Cap_F: capacité d'envoi d'un fournisseur F
u_{ij}: capacité de l'edge ij
T: Durée maximale acceptable de transbordement
Variables:
X_{ij}: Nombre de paquets qui transitent de i vers j
Y_{ij}: Variable binaire, vaut 1 si X_{ij} est supérieur à 0, 0 sinon
Z_i: Variable binaire, vaut 1 si la plateforme i est utilisée, 0 sinon
Fonction Objectif:
La fonction objectif minimise les coûts de transfert des marchandises.
Contraintes:
- Ne pas dépasser la capacité maximale par client
- Ne pas dépasser la capacité maximale par fournisseurs
- Ne pas dépasser la capacité maximale des edges
- Respecter le temps d'execution

2 Modèle Mathématique

Fonction Objectif:
$\min z_c = \sum_{i=1}^{F}(\sum_{j=1}^{P}(\sum_{k=1}^{D}(X_{ij}\times h_{ij}+X_{jk}\times h_{jk}+Y_{ij}\times c_{ij}+Y_{jk}\times c_{jk}+Y_j\times c_j)))$
Contraintes:
$\sum_{i=1}^{i=F}(\sum_{j=1}^{j=P}(X_{jk})) = Cap_D$ pour tout k entier dans [1;D]
$\sum_{j=1}^{j=P}(\sum_{k=1}^{k=D}(X_{ij})) = Cap_F$ pour tout i entier dans [1;D]
Pour chaque i,j: $X_{ij} <= u_{ij}$
$\sum_{i=1}^{F}(\sum_{j=1}^{P}(\sum_{k=1}^{D}(t_{ij} \times X_{ij} + s_j + t_{jk} \times X_{jk}))) <= T$

3 Algorithme de résolution

Nous avons décider d'utiliser le language JAVA pour des raisons de facilité et de portabilité.

Afin de résoudre ce problème, nous avons préalablement choisi l'algorithme du Branch and Bound qui nous permettait de trouver en théorie une solution optimale au bout d'un certain temps.

Dans un second temps, après avoir discuté un peu autour de nous, nous avons vu que beaucoup de personnes allaient utiliser cet algorithme, nous avons donc décidé de changer de méthode afin de fournir un travail original et un challenge plus important (étant donné que le Branch and Bound a une efficacité prouvée, nous voulions voir si un autre algorithme aurait pu concurrencer le Branch and Bound).

Nous avons remarqué que ce problème de transbordement pouvait se rapporter à un problème de flux maximal à coût minimal (étant donné qu'il faut faire transiter une ressource d'un point A vers un point B) donc nous nous sommes concentrés sur l'implémentation de cet algorithme. Nous avons donc commencé par reconstruire un algorithme de lecture des fichiers de données, qui nous a permis d'implémenter le programme de la manière qui nous semblait la plus simple en terme de structures (objets et méthodes), puis nous avons crée la solution initiale et enfin nous avons implémenté notre algorithme afin de répondre aux problèmes proposés.

4 Tests et résultats

Voici le tableau récapitulatif de nos tests avec la version stable de notre programme, on y retrouve le nom du fichier utilisé, le nombre de nodes et d'edges, le temps d'exécution, la solution trouvée et le nombre de paquets restants (dans le cas d'un algorithme avec un problème de priorité).

Nom du Modèle	Nb Nodes	Nb Edges	Tps exécution	Solution	Paquets restants
tshp10-01	10	9	0.0	3938	0
tshp010-01	10	21	0.0	4856	0
tshp10-02	10	16	0.0	2990	0
tshp010-02	10	21	0.0	2101	0
tshp010-03	10	16	0.0	1949	0
tshp010-04	10	16	0.0	2110	0
tshp010-05	10	16	0.0	2864	0
tshp020-01	20	75	0.0	4106	0
tshp020-02	20	91	0.0	4902	0
tshp020-03	20	75	0.0	5163	0
tshp020-04	20	75	0.0	3854	0
tshp020-05	20	36	0.0	11120	0
tshp050-01	50	576	0.0	13671	18
tshp050-02	50	504	0.0	15147	31
tshp050-03	50	301	0.0	9836	0
tshp050-04	50	525	0.0	13533	0
tshp050-05	50	301	0.0	12956	0
tshp100-01	100	979	6.0	24992	25
tshp100-02	100	1275	7.0	30459	5
tshp100-03	100	1476	6.0	28657	107
tshp100-04	100	2275	9.0	19066	119
tshp100-05	100	2275	9.0	23289	102

5 Avantages, Inconvénients et Problèmes

Concernant les inconvénients de cet algorithme, on peut noter que c'est un algorithme de construction, donc la contrainte de temps d'exécution du programme est un problème majeur étant donné que si on interrompt le programme avant qu'il ait fini sa construction, on a pas de solution. Deuxième problème à noter, le temps d'exécution en lui même. En effet, plus il y aura de noeuds dans le problème, plus l'algorithme va prendre de temps pour résoudre le problème (comme on peut le voir dans le tableau ci-dessus, on a des temps qui vont jusqu'à 9 secondes pour un modèle à 100 noeuds, et un modèle à 500 noeuds dure plus de 3h).

Puis, concernant les avantages, notre algorithme est très efficace en terme de temps quand il s'agit de traiter un problème avec peu de noeuds (comme on peut le voir dans le tableau ci-dessus, il faut aller à un problème à 100 noeuds pour obtenir un temps d'exécution notable). Ensuite, comme c'est un algorithme de construction, il trouve directement la meilleure solution optimale, donc la qualité de la solution est meilleure qu'avec d'autres algorithmes. C'est également un algorithme qui va toujours trouver la solution optimale peut importe le nombre de noeuds.

Et enfin, nous allons parler des problèmes que nous avons rencontré ainsi que des solutions que nous avons mises en place pour y palier.
Concernant le problème de la contrainte de temps qui induit de ne pas trouver de solution si le temps est trop court, nous avons décidé de mettre en place une solution initiale (comme dans les algorithmes d'optimisation ne reposant pas sur

une construction) afin d'avoir une solution quel que soit le temps d'execution du programme. Dans ce cas, la solution est considérée comme binaire: soit on obtient une solution initiale par défaut, qui n'est pas optimisée (mais qui est une solution tout de même), soit on obtient une solution optimale. Un autre problème que nous avons rencontré est le problème de la construction en elle même. En effet, l'algorithme parcourt tous les chemins viables et choisit le plus optimal pour faire passer des paquets. Mais il ne prend pas en compte le fait qu'un chemin peut ne mener qu'à un seul client, et si ce client est déjà satisfait par un autre fournisseur, ce chemin sera bloqué et des paquets pourront ne pas être transférés.
Afin de contrer ce problème, nous avons essayé de détecter ces cas de figure afin de remplir d'abord les chemins uniques, et ensuite les chemins optimaux, afin que tous les paquets puissent partir. Nous avons remarqué que ce problème de blocage était présent dans les problèmes à 50 noeuds et plus (correspondant dans le tableau aux lignes ou il y a des paquets restants). Etant donné que les chemins sont parcourus les uns après les autres, il est logique qu'il y ait des problèmes. Pour améliorer notre programme, nous avons donc pensé a implémenter une chaîne améliorante, adaptée au projet, qui prendrait en compte les chemins uniques et en qui implémenterai un système de priorités sur ces chemins. Malheureusement face à la complexité du problème, nous avons été dans l'incapacité de mener à bout cette solution.
Et enfin, à la fin de l'implémentation de l'algorithme, nous avons voulu remettre en place la solution initiale (que nous avions enlevée pour des raisons pratiques), mais il y a eu des conflits par rapport au graph créé avec l'algorithme. Nous avons donc décidé de ne pas mettre de solution initiale (elle n'aurait fait que ralentir l'algorithme étant donné qu'il aurait fallu recréer tout un graph a part avec notre structure de données actuelle) et ainsi obtenir une solution certes incomplète parfois, mais tout de même optimale.