École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise

Projet de recherche opérationnelle : Optimisation dans les réseaux optiques

Constant Gayet et William Delaisse

13 janvier 2023

Sommaire

1 Préface

1.1 Introduction

Ce projet a pour but de travailler sur un problème de recherche opérationnelle qui s'inscrit dans un contexte d'acheminement de l'information dans les réseaux optiques. Dans le contexte actuel, l'acheminement fiable et à moindre coût de l'information dans un réseau urbain de plus en plus dense et interconnecté est au cœur des préoccupations. Le projet proposé se concentre sur les problèmes de recherche de structure de recouvrement d'un réseau modélisé par un graphe sous contrainte sur le degré des sommets. L'arbre étant la structure la plus utilisée dans le domaine des réseaux (l'introduction du broadcast/multicast a augmenté son impact), nous cherchons donc à maximiser la robustesse du réseau, qui dépend du degré des sommets de l'arbre, ce qui correspond à couvrir les sommets de l'arbre en utilisant un minimum de liens. Par ailleurs, il est important de limiter le nombre de duplications de message pour assurer un routage efficace et de limiter le degré des sommets pour assurer la robustesse du routage.

1.2 Structure du projet

Le projet est consitué des fichiers suivants :

- Le fichier *main.py* est l'executable qui permet de se servir proprement des fonctions implémentant tous nos programmes. En effet, grâce au fichier *utils.py*, nous pouvons écrire de manière automatisée les résultats aux tests.
- Les fichiers plnecp.py et plnecpm.py sont respectivement les implémentations des deux premiers programmes linéaires (Formulation basée sur le concept de flot et sur le concept de flots multiples)
- Les fichiers plnecp.pl et plnecpm.pl sont les fichiers décrivant les programmes linéaires créés par la librairie python Pulp.
- Le fichier heuristiques.py contient l'implémentation des trois heuristiques décrites dans le sujet, à savoir :
  - Une heuristique basée sur le poids des sommets
  - Une heuristique basée sur la couleur des commets
  - Une heuristique basée sur une combinaison des deux heuristiques précédentes

2 Programme linéaire basé sur le concept de flot (PLNECP)

Le premier algorithme que nous avons implémenté est un programme linéaire basé sur le concept de flot. Dans cette version de l'algorithme, on considère deux variables qui correspondent aux différentes arêtes et sommets et qui valent 1 si l'arête ou le sommet est sélectionné et 0 sinon. Une des fonctions objectives assure que le graphe résultant est sans cycle tandis qu'une autre fonction objective assure que le graphe résultant ne contient que n-1 arêtes et donc, est connexe ce qui nous donne bien un arbre.

3 Programme linéaire basé sur le concept de flots multiples (PLNECPM)

Le second algorithme que nous avons implémenté est un programme linéaire basé sur le concept de flots multiples. Dans cette version, on considère la version orientée symétrique du graphe et on ajoute à chaque arc une quantité de flots. L'objectif de l'algorithme est de trouver les arcs faisant transiter une quantité de flots non nulle car ce sont ces arcs qui correspondent aux arêtes de l'arbre de recouvrement. L'objectif de cet algorithme est identique à celui du modèle précédent.

4 Un approximation nécessaire pour améliorer la complexité grâce à des heuristiques

4.1 Heuristique basée sur le poids des arêtes

La première heuristique est basée sur le poids des arêtes. Pour cette heuristique, on ajoute une fonction qui attribue un poids à chaque arête qui indique la probabilité que cette arête crée un sommet de branchement dans l'arbre de recouvrement. À chaque tour de l'algorithme, on sélectionne une arête de poids minimum qui n'est pas déjà dans l'arbre et on réduit les chances de sélectionner ses voisins afin d'éviter les cycles. L'algorithme s'arrête quand n-1 arêtes sont sélectionnées ce qui assure que le graphe est connexe.

4.2 Heuristique basée sur la couleur des sommets

La seconde heuristique est basée sur l'assignation d'une couleur à chaque sommet. La couleur d'un sommet permet de déterminer si ce sommet deviendra un sommet de branchement quand une arête incidente à ce sommet est sélectionnée pour être insérée dans l'arbre. L'algorithme choisit ainsi une arête ayant un nombre minimum d'extrémités de haut degré. Afin de minimiser le nombre de nouveaux sommets à haut degré potentiels, l'algorithme sélectionne, parmi les arêtes ayant un nombre minimum d'extrémités de degré 2, celles ayant le

moins de sommets de degré 1 comme extrémités.L'algorithme s'arrête quand n-1 arêtes sont sélectionnées comme pour l'algorithme précédent.

4.3 Heuristique basée sur une combinaison des deux précédentes

La troisième heuristique est basée sur une combinaison des deux approches précédentes ce qui permet d'obtenir des résultats plus précis. À chaque itération, on choisit en priorité les arêtes de poids minimum puis on utilise le principe utilisé par l'heuristique basée sur la couleur des sommets.

5 Résultats obtenus et interprétation

5.1 Résultats sur des graphes relativement petits (20-500 noeuds)

En un temps de calcul raisonnable, nous arrivons à obtenir des résultats pour les graphes de 20 à 500 noeuds avec le programme linéaire basé sur le concept de flot. En effet, le calcul de 66 instances sur les 400 fournies nous a pris environ 4h. Nous pouvons comparer cela à l'heuristique basée sur la couleur des sommets, qui a pris seulement 36s pour les mêmes instances. Cependant, nous pouvons constater que les résultats fournis par l'heurisitque **ne permettent pas de fournir une solution optimale**. Etudions l'exemple d'un graphe de 40 noeuds (*Spd_RF2_40_50_619.txt*). Voici le graphe original :

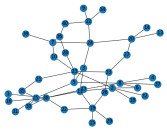

FIGURE 1 – Graphe de 40 noeuds donné commme instance du problème

Le résultat que nous fournit le programme linéaire ici est correct : le graphe est connexe et le nombre de noeuds de branchements paraît minimal (seulement 7 noeuds de branchement sur cet arbre).

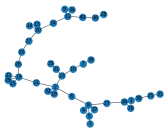

FIGURE 2 – Résultat exact grâce au programme linéaire basé sur le concept de flot

Nous pouvons également voir que notre heuristique de coloration de graphe est inexacte puisqu'elle fournit un graph non connexe.

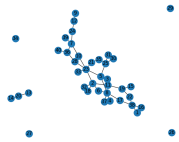

FIGURE 3 – Résultat par calcul à l'aide d'une heuristique de coloration de noeuds

5.2 Résultats sur des graphes relativement grands (500-1000 noeuds)

Nous n'avons pour cette section aucun résultat exact exploitable. En effet, les instances sont trop grosses pour être caculées sur nos machines. Nous avons, par exemple, effectué un essai sur un graphe de 1000 noeuds. Après un calcul de 2h et un fichier de calcul de plus de 2Go, nous avons stoppé le script.

6 Difficultés rencontrées et limites

6.1 Difficultés rencontrées

Tout d'abord, nous n'avions jamais travaillé avec les librairies python Pulp et NetworkX. Les comprendre et les utiliser constituait déjà une première tâche. En effet, nous n'étions jamais passé du côte "pratique" des programmes linéaires. Nous avions, dès lors, seulement eu une approche théorique sur les programmes linéaires, les appliquer à un cas concret était alors nouveau.

Ensuite, comme nous n'étions que deux, notre charge de travail était plus conséquente. Par ailleurs, notre compréhension des heuristiques a longtemps été très mauvaise et nous avons eu beaucoup de mal à les comprendre et à les implémenter, ce qui nous a fait perdre beaucoup de temps.

6.2 Limites

Tout d'abord, nous sommes conscients des limites de notre code et de notre travail de manière plus générale. En effet, nous n'avons pas implémenté la **Formulation de Martin** par manque de temps et par priorisation du travail : nous voulions d'abord être capables d'implémenter et de faire fonctionner les heuristiques plutôt que refaire un travail similaire à l'implémentation des deux premiers programmes linaires.

Par ailleurs, lorsque nous essayons de faire fonctionner un de nos programmes linéaires sur une grosse instance, le fichier de calcul prends une taille colossale et nous n'avons pas réussi à obtenir un résultat en plus d'une heure sur une instance de 1000 nœuds. Cela est normal en considérant la complexité des problèmes mais nous n'avons cependant pas de résultat sur de grosses instances.

7 Conclusion

Tout d'abord, ce projet a représenté pour nous une grande partie de documentation et de debogage. En effet, nos connaissances restreintes en python, particulièrement sur les librairies utilisées, ne nous ont permis de produire un travail de bonne qualité. Par ailleurs, nous avons passé une partie conséquente du temps de travail à essayer de comprendre le fonctionnement des heuristiques, qui ne nous était aucunement familier. Malgré cela, nous avons réussi à obtenir des résultats théoriques sur des instances de taille raisonnable (< 500 noeuds). Leur interprétation est cependant limitée puisqu'il nous est difficile de comparer nos résultats et nos performances avec des heuristiques incertaines.