Rapport du Projet d'Optimisation

Guillaume Aubian

9 janvier 2016

Table des matières

Première partie
Outils Utilisés

Pour ce projet, j'ai décidé d'utiliser OCaml, en essayant de faire le plus de fonctionnel, pour la beauté du défi, même si malheureusement ça a beaucoup compliqué la tache pour le parsing, sans compter qu'il a fallu coder à partir de zéro une librairie pour les graphes et pour les matrices.

Tout le projet est de moi, à l'exception du Makefile, qui est le Makefile usuel pour OCaml et qui vient de l'INRIA, ainsi que des instances proposées comme exemples, qui sont toutes tirées de TSPlib.

Deuxième partie
Comment l'utiliser

Il faut tout d'abord télécharger le projet, disponible à l'adresse :
`https://github.com/gaubian/DM_Optimization`

Une fois dans le dossier correspondant, il faut le compiler en tapant «make» dans la console.

Enfin, on lance le programme avec la commande :

./project x input_file output_file

Où $x \in \{naive\_tour; approximate\_tour; naive\_LP; cut\_LP\}$

Ceci aura pour effet de prendre le graphe situé dans input_file, de lui appliquer x et d'écrire le résultat dans output_file.

Les graphes doivent être donnés dans le format décrit par la documentation de TSPlib trouvable ici :

`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`

Troisième partie
Esprit Général du projet

J'ai découpé mon projet en une multitude de modules : un qui s'occupe des opérations matricielles, un des graphes, un consacré au simplexe, un au parsing, un à l'algorithme de calcul de la borne minimale, et enfin Main, qui organise le tout.

En réalité, ces modules sont plutôt des foncteurs, qui prennent en argument le corps dans lequel on veut faire nos calculs. Par exemple, on peut vouloir le faire sur des réels (représentés par des flottants), des rationnels (des paires d'entiers, pour ne pas avoir d'arrondi), sur le corps des rationnels multiplié par la racine carrée d'un entier (ce qui permet à la fois de ne pas avoir d'arrondi, tout en gérant la racine carré qui peut itervenir lors du calcul de la fonction distance), sur les rationnels mais représentés par des entiers sur un nombre de bits arbitraire etc. (je n'ai implémenté que les deux premières possibilités).

Grace aux foncteurs, il suffit juste de modifier le corps en question, ce qui se fait très facilement en changeant le fichier which_structure.ml, en remplaçant FloatField par RationalField par exemple (les définitions des structures sont disponibles dans structures.ml).

Quatrième partie
Le Simplexe

Assez facilement la partie la plus compliquée du projet, entre le fait qu'il fallait rester sur de l'impératif et l'absence totale des matrices en OCaml.

Ma stratégie a été dans un premier temps de coder une librairie matricielle la plus complète possible (avec notamment des fonctions qui ne servent à rien dans le projet), après quoi j'ai pu coder le simplexe.

Sur les deux choix de pivot que j'ai codés, j'ai décidé de poser par défaut celui du minimum lexicographique, jugeant que c'était plus simple que de devoir passer le choix de l'utilisateur de l'interface (Main) au module low-level qu'est Simplex, pour une fonctionnalité pas très importante.

Cinquième partie
Les Graphes

Tout comme pour les matrices, j'ai décidé de coder un module pour les graphes, si ce n'est que cette fois-ci, la structure est totalement pure, et le module n'utilise que du fonctionnel.

Étrangement, j'ai trouvé beaucoup plus facile de coder Held-Karp que l'algorithme naïf, et c'est donc lui qui tient lieu d'algorithme naïf quand l'utilisateur le demande.

Pour l'heuristique, j'ai décidé d'appliquer l'algorithme suivant, en partant d'un tour réduit à un noeud.

1. Si deux sommets consécutifs ont un voisin commun à l'extérieur du tour, je peux éventuellement «casser» l'arête entre ces deux sommets,

et intégrer le nouveau sommet en question. Parmi tous les sommets que je peux ainsi intégrer, je m'occupe en priorité de ceux qui augmentent le moins la longueur du cycle.

2. Si ce n'est plus possible, je considère le sommet externe au tour le plus proche d'un sommet du tour, et je fais faire un aller retour de ce dernier vers le nouveau sommet.
3. Si ce n'est pas possible c'est que mon tour recouvre le graphe, et j'ai donc ma solution.

Sur des petites instances, il trouve quasiment toujours l'optimal. Sur gr48, un graphe à 48 sommets, il trouve 6239, contre 5046 pour l'optimal, et un peu moins de 80000 contre presque 60000, pour pr144, à 144 sommets. Mais toujours en moins d'une seconde.

Sixième partie

Programme Linéaire et garantie d'approximation

Dans cette partie, j'ai vraiment senti que le fait d'avoir codé les librairies pour les graphes et les matrices était d'une aide précieuse, et elle a été relativement facile à coder.

L'algorithme exhaustif est excessivement mauvais, et il est lent pour des graphes à seulement 9-10 noeuds.

Pour le calcul de la coupe minimum, j'utilise l'algorithme de Stoer-Wagner, qui fonctionne comme suit :

Supposons que je puisse, étant donné un graphe, trouver deux sommets s, t et une st-coupe minimale A. Alors, considérons B, une coupe minimale de G :

— Soit s et t sont séparés par B, et donc A est aussi une coupe minimale G
— Soit ce n'est pas le cas, auquel cas je peux fusionner s et t en un sommet s', qui récupère les arêtes de s et t (et somme les poids vers les voisins communs à s et t). Le nouveau graphe obtenu a alors un noeud de moins que G, et on peut déduire aisément de sa coupe minimale une coupe minimale de G

On obtient donc notre récursion, le cas de base étant trivial.

En plus d'être esthétique, l'algorithme est relativement rapide sur des instances testées à la main. Malheureusement, les graphes de TSPlib étant complets, et cet algorithme subissant très mal un nombre élevé d'arêtes, on ne peut l'appliquer sur aucune instance de TSPlib (la plus petite a 122 arêtes, sachant que le simplexe est théoriquement (et dans le pire des cas, qui n'arrive que rarement) exponentiel en le nombre d'arêtes...).

Septième partie

Nombre de variables

Ce soucis devrait normalement être réglé dans cette partie, mais je n'arrive malheureusement pas bien à la comprendre.

En effet, si je comprends que les contraintes correspondent à des ensembles d'arêtes, que si on sait qu'une contrainte est non-satisfaite, on peut rajouter à E' les arêtes concernées par la contrainte mais pas par E', je ne comprends pas comment déduire de la valeur associée à une contrainte (car c'est bien ce que nous renvoie le dual, un flottant par contrainte) qu'elle est satisfaite ou non ? Je pense qu'il faut montrer que ce PL (dual) est intégral, mais je n'y arrive pas.

Huitième partie

Questions :

1 Question 1 :

Si on considère un graphe complet à 4 sommets, où chaque arête est pondérée par 1, alors on remarque que mettre les x_e tous à $\frac{2}{3}$, résout notre programme linéaire. Donc notre programme linéaire n'est pas intégral

2 Question 2 :

Étant donné un cycle du graphe ne passant pas pour tous les sommets, mais seulement par $c \neq |V|$ sommets, on peut dire que la somme des x_e sur ce cycle est inférieure ou égale à n-1.

On peut aussi dire que la somme totale des x_e du graphe fait |V| - 1.