M1 ALMA/ATAL/ORO — Université de Nantes
X8II110 - Data structures and Algorithms — 2016/2017

Feuille de Travaux Pratiques
Algorithmes de Résolution de VERTEX COVER

Mars-Avril 2016

Six séances de TP seront consacrées à ce projet, lequel est à effectuer soit seul(e), soit en binôme.

La programmation se fera dans le langage de votre choix.

A l'issue des 6 séances, vous rendrez les sources de votre projet, sous la forme d'une archive NOM1-NOM2.zip contenant votre code, ainsi que votre compte-rendu. La décompression de l'archive doit produire un répertoire NOM1-NOM2 contenant tous les éléments de votre travail (compte-rendu, sources, exécutable, jeux d'essai le cas échéant) et un fichier texte contenant les instructions de compilation et d'exécution.
Assurez-vous que vos programmes compilent et fonctionnent sous Linux dans les salles machine du CIE.

Cette archive est à déposer sous Madoc. La date limite de restitution est fixée au Lundi 10 Avril 2017 à 12h (aucun dépôt après cette heure ne sera possible).

Un malus pourra être appliqué, en fonction du non-respect de tout ou partie des consignes ci-dessus.

1 Présentation succincte

Dans ce projet, il vous est demandé d'implémenter et de tester plusieurs algorithmes de résolution du problème VERTEX COVER, à savoir :

1. Algorithme GLOUTON-VC : une variante de l'algorithme de 2-approximation vu en CM, dans laquelle à chaque itération l'arête est choisie arbitrairement *parmi les arêtes* (u, v) *telles que la somme des degrés* $d(u) + d(v)$ *est maximum*.
2. Algorithme IPL-VC : correspond à l'algorithme de 2-approximation vu en CC (voir dernière page).
3. Algorithme ARB-VC : l'algorithme de type "Arbre de Recherche Borné" vu en CM, et décrit succinctement ici. Tant qu'il existe un sommet u de degré ≥ 3, tester récursivement les deux cas possibles : (1) u est dans le vertex cover et on supprime u de G et (2) u n'est pas dans le vertex cover, tous ses voisins y sont, et on supprime ces voisins de G. Quand le processus s'arrête, si le graphe contient encore des arêtes, alors il se compose de cycles et de chemins, et dans ce cas la taille du vertex cover peut se calculer facilement.
4. Algorithme KERNEL-VC : l'algorithme de Kernelization vu en CM.

2 Travail demandé

Le but de ce projet est de comparer les résultats renvoyés par ces quatre algorithmes, pour des graphes de tailles différentes, générés aléatoirement. Il est donc demandé de répondre aux questions suivantes :

1. Implémenter les quatre algorithmes indiqués dans la section précédente.
2. Pour un grand nombre de valeurs de n, exécuter chacun de ces algorithmes sur des graphes G à n sommets, dont les arêtes sont générées aléatoirement à partir d'un paramètre p (la méthode est expliquée plus loin). Pour chaque valeur de n, on remplira le tableau suivant (m désigne le nombre d'arêtes, Δ le degré maximum, et d_{moy} le degré moyen de G) :

$n =$	m	Δ	d_{moy}	KERNEL-VC Temps	Val	Nb Exec	ARB-VC Temps	Val	Nb Exec	GREEDY-VC Temps	Val	IPL-VC Temps	Val
$p = 4/n$													
$p = 5/n$													
$p = (\log n)/n$													
$p = 1/\sqrt{n}$													
$p = 0.1$													
$p = 0.2$													

3. Les tests doivent être reproductibles. Il faut donc proposer un menu à l'utilisateur, qui lui demande une valeur pour n, et qui affiche à l'écran le tableau ci-dessus pour la valeur de n fournie. De plus, sans sortir du programme, l'utilisateur devra pouvoir tester d'autres valeurs de n.
4. Analyser l'ensemble des résultats obtenus, et en particulier répondre aux questions suivantes :
   (a) Reproduire dans le rapport chaque tableau de résultats (*un tableau par valeur de* n).
   (b) Quelle est la complexité de chacun des quatre algorithmes implémentés ?
   (c) Pour chacun des quatre algorithmes, et pour chaque type de graphe (parmi les six proposés), quelle est la plus grande valeur de n que vous avez pu traiter en 3 minutes ? Donner la réponse sous la forme d'un tableau.
   (d) Pour chaque valeur de p, donner le ratio d'approximation moyen de GREEDY-VC. Cela ne peut se faire que pour les valeurs de n pour lesquelles l'optimal est obtenu (par KERNEL-VC ou par ARB-VC).
   (e) Même question concernant le ratio d'approximation moyen d'IPL-VC.
   (f) Au vu de l'ensemble de vos résultats, quel est le meilleur algorithme à utiliser, en fonction de n, p, Δ et d_{moy} ? Votre réponse doit être détaillée, et doit s'appuyer à la fois sur (a) la qualité du résultat et (b) le temps d'exécution.

Précisions :

— La génération aléatoire de G, une fois n et p fixés, se réalise comme suit : pour toute paire de sommets u et v de G, avec $u \neq v$, la probabilité que l'arête u, v existe est égale à p.

— Pour chaque valeur de n fixée, on générera six graphes G, avec les valeurs de p suivantes : $4/n$, $5/n$, $(\log n)/n$, $1/\sqrt{n}$, 0.1 et 0.2.

— Les valeurs de n à tester sont les suivantes :
  — de 10 en 10 jusque 100 (donc $n = 10, 20 \ldots 100$), puis
  — de 20 en 20 jusque 500, puis
  — de 50 en 50 jusque 1000, puis
  — de 100 en 100

— Pour chacun des cinq algorithmes :
  — "Temps" désigne le temps d'exécution sur le graphe testé. Il sera exprimé en secondes, avec 3 chiffres significatifs.
  — "Val" désigne la taille du Vertex Cover V' trouvé par l'algorithme, c'est-à-dire le nombre de sommets que contient V'.
  — Pour les algorithmes KERNEL-VC et ARB-VC, "Nb Exec" désigne le nombre d'exécutions qui ont été nécessaires pour déterminer la bonne valeur de k (voir ci-dessous).

— Les deux algorithmes exacts, KERNEL-VC et ARB-VC, ont besoin du paramètre k, inconnu a priori. Pour chacun de ces deux algorithmes, plusieurs exécutions sont donc nécessaires pour connaître la valeur optimale.
  Pour restreindre l'intervalle de recherche de la valeur optimale, on pourra exécuter d'abord GREEDY-VC, qui renverra une valeur v, et IPL-VC, qui renverra une valeur v'. On appelle $w = \min\{v, v'\}$. On sait alors que le k optimal est dans l'intervalle $[\frac{w}{2}; w]$, et il est donc possible d'effectuer une recherche dichotomique dans cet intervalle.

— Pour une valeur de n donnée, il est possible qu'un (ou plusieurs) des algorithmes soit "trop long(s)" à exécuter – on s'attend à ce que ce soit le cas pour KERNEL-VC et ARB-VC, qui sont exponentiels en k. Pour un graphe G (où n et p sont fixés), on autorisera un temps maximum de 3 minutes pour qu'un algorithme réponde. Si ce temps est dépassé, on indiquera NRP (pour "Ne Répond Pas") dans la case du tableau qui correspond. Si un dépassement mémoire se produit, on indiquera DM dans le tableau.

Exercice 1 (Programmation Linéaire et Approximation pour MIN-VC)

Voici la description d'un programme linéaire en nombre entiers (qu'on appellera (IP)), construit à partir d'un graphe G :

$$\begin{aligned} &\text{minimize } x_1 + x_2 + \ldots + x_n \\ &\text{subject to } x_i + x_j \geq 1 \qquad \forall (v_i, v_j) \in E(G) \\ &\qquad\qquad x_i \in \{0, 1\} \qquad\qquad \forall v_i \in V(G) \end{aligned}$$

1. Indiquer comment, partant d'une solution de (IP), on peut construire une solution de MIN-VC qui a le même optimum (on attend ici une réponse précise et argumentée).
2. Indiquer comment, partant d'une solution de MIN-VC, on peut construire une solution de (IP) qui a le même optimum (on attend ici une réponse précise et argumentée).

Les Questions 1. et 2. ci-dessus montrent donc que les problèmes (IP) et MIN-VC sont équivalents.
Voici maintenant la description d'un programme linéaire (qu'on appellera (LP)). Cela veut dire qu'ici les y_i ne sont pas nécessairement des entiers, mais *peuvent prendre n'importe quelles valeurs réelles entre 0 et 1*.

$$\begin{aligned} &\text{minimize } y_1 + y_2 + \ldots + y_n \\ &\text{subject to } y_i + y_j \geq 1 \qquad \forall (v_i, v_j) \in E(G) \\ &\qquad\qquad 0 \leq y_i \leq 1 \qquad\qquad \forall v_i \in V(G) \end{aligned}$$

Les problèmes de type (LP) peuvent se résoudre en temps polynomial, alors que les problèmes de type (IP) sont NP-complets. L'idée est donc d'utiliser le problème (LP) pour obtenir une approximation du problème (IP), et donc une approximation du problème MIN-VC.

3. Montrer que, pour tout graphe G, $opt(LP) \leq opt(IP)$.

Appelons $y^* = (y_1^*, y_2^* \ldots y_n^*)$ une solution optimale obtenue pour (LP). On définit alors $x^A = (x_1^A, x_2^A \ldots x_n^A)$ de la manière suivante : pour tout $1 \leq i \leq n$, $x_i^A = 1$ si $y_i^* \geq \frac{1}{2}$, et $x_i^A = 0$ sinon.

4. Montrer que si y^* est une solution optimale pour (LP), alors x^A est une solution pour (IP) (cela revient à montrer que si $y_i^* + y_j^* \geq 1$, alors $x_i^A + x_j^A \geq 1$).
5. Montrer que pour tout $1 \leq i \leq n$, $x_i^A \leq 2y_i^*$.

Soit $opt(LP) = y_1^* + y_2^* + \ldots y_n^*$, et $sol(IP) = x_1^A + x_2^A + \ldots x_n^A$.

6. Montrer que $sol(IP) \leq r \cdot opt(LP)$, où r est une valeur à déterminer.
7. Conclure quant à l'approximabilité de (IP) et de MIN-VC.
8. Indiquer (en français) les grandes étapes de l'algorithme d'approximation de ratio r pour MIN-VC que nous venons d'étudier.