Comparaison d'algorithmes pour le problème du Vertex Cover sur de grands graphes*

Eric Angel[1], Romain Campigotto[1] et Christian Laforest[2]

[1] Laboratoire IBISC ; Université d'Evry ; Tour Evry 2, 523 place des terrasses, 91000 Evry, France
{angel,campigotto}@ibisc.univ-evry.fr
[2] LIMOS ; Université B. Pascal ; Campus scientifique des Cézeaux, 63173 Aubière cedex, France
christian.laforest@isima.fr

Mots-Clés : *grands graphes, Vertex Cover, analyse en moyenne.*

Introduction. Lorsque l'on traite de façon classique un problème d'optimisation sur un graphe, celui-ci est le plus souvent disponible dans sa totalité sur la machine de traitement. Il peut être modifié, mis à jour (les sommets peuvent être marqués, les arêtes supprimées, *etc.*) et la solution peut être conservée en mémoire. Cependant, de plus en plus d'applications produisent des quantités de données qui sont trop importantes et qui ne peuvent pas être stockées ni traitées dans ce modèle.

Nous nous intéressons ici au traitement d'un problème NP-Complet de minimisation sur de grands graphes, celui du *Vertex Cover*. La difficulté est double ; à la NP-Complétude intrinsèque (voir [4]) est ajoutée la difficulté de manipuler le graphe avec des restrictions strictes :

1. Le graphe fourni en entrée ne doit pas être modifié (intégrité de l'instance).
2. Le graphe doit être « scanné » morceau par morceau car il est trop conséquent pour être chargé dans sa totalité.
3. Du fait de sa grande taille potentielle, le résultat doit être envoyé « à la volée » dans une mémoire externe dès qu'un nouveau sommet est ajouté.

Par conséquent, on suppose que la taille et la provenance du graphe imposent de sévères contraintes pour son traitement. Pour préciser les choses, dans la plupart des applications réelles, les sommets possèdent des *labels* (ou étiquettes) qui sont 2 à 2 distincts et qui peuvent être ordonnés (par ordre lexicographique par exemple). Cette notion importante intervient dans les algorithmes que nous allons décrire.

Des algorithmes adaptés. Il existe beaucoup d'algorithmes d'approximation pour le problème du *Vertex Cover* (voir [2]), mais la plupart d'entre eux ne respectent pas les contraintes de ce modèle. Nous avons étudié et comparé 3 algorithmes inspirés par *D. Avis et T. Imamura* [3] : ListLeft, Sorted-LL et Anti Sorted-LL (voir [1]). L'algorithme ListLeft fonctionne de la façon suivante : pour chaque sommet u scanné, u est mis dans la solution si et seulement si il possède un voisin ayant un label plus grand que lui. L'algorithme Sorted-LL (*resp.* Anti Sorted-LL) procède de la manière suivante : pour chaque sommet u scanné, u est mis dans la solution si et seulement si il possède un voisin de degré inférieur (*resp.* supérieur) ou s'il possède un voisin de même degré ayant un label plus grand (*resp.* plus petit) que lui.

*Travail partiellement soutenu par les projets *ToDo* (ANR) et « *Approximation rapide* » du GDR-RO.

Comparaison des algorithmes[1]**.** Si l'on en reste à une analyse classique en pire cas, Sorted-LL a un meilleur rapport d'approximation (de $\frac{\sqrt{\Delta}}{2} + \frac{3}{2}$) que ListLeft et Anti Sorted-LL (qui sont en $\mathcal{O}(\Delta)$). Nous ne nous sommes pas contentés de résultats en pire cas et nous avons comparé ces algorithmes en moyenne. Pour un étiquetage donné, ils sont déterministes. Cependant, pour un graphe à n sommets, il existe exactement $n!$ ordres induits par les labels : l'exécution des algorithmes à partir d'étiquetages distincts peut donner des résultats différents. Par exemple, nous avons montré que l'algorithme ListLeft a la particularité de pouvoir retourner, pour n'importe quel graphe, une solution optimale ou une solution de taille $n-1$ (s'il s'agit d'un graphe connexe).

Nous avons déterminé de façon exacte les espérances de chacun des 3 algorithmes (voir [1]). Les 3 formules obtenues sont similaires. A titre d'illustration, celle de ListLeft est : $n - \sum_{u \in V} \frac{1}{d(u)+1}$. Notons que pour n'importe quel graphe, ces espérances peuvent se calculer en temps polynomial. Nous avons exhibé de nombreuses familles de graphes pour lesquelles Sorted-LL possède des performances en moyenne meilleures que les 2 autres (voir [1]). Cependant, comme le résume le tableau 1, il existe des familles de graphes pour lesquelles, en moyenne, ListLeft est meilleur et d'autres pour lesquelles, en moyenne, Anti Sorted-LL est meilleur.

Nous avons aussi montré qu'à partir d'un même étiquetage, Sorted-LL retournait toujours une solution strictement meilleure que Anti Sorted-LL sur des chemins.

Pour résumer, nous avons étudié 3 algorithmes qui sont compatibles avec notre modèle. Cependant, cette étude ne nous permet pas de donner de façon stricte un gagnant parmi les 3. A l'avenir, il faudra notamment identifier de manière plus précise les catégories de graphes pour lesquelles tel ou tel algorithme est le meilleur.

Algorithmes / **Familles de graphes**	ListLeft	Sorted-LL	Anti Sorted-LL
Graphes réguliers	1^{er} *ex aequo*	1^{er} *ex aequo*	1^{er} *ex aequo*
Etoiles ou chemins	$2^{ème}$	1^{er}	$3^{ème}$
Graphes bipartis complets	$3^{ème}$	1^{er}	$2^{ème}$
Graphes d'*Avis-Imamura*	$3^{ème}$	1^{er} *ex aequo*	1^{er} *ex aequo*
Graphes d'*Avis-Imamura* étendus (version *ASL*)	$3^{ème}$	$2^{ème}$	1^{er}
Graphes d'*Avis-Imamura* étendus (version *ALS*)	$2^{ème}$	$3^{ème}$	1^{er}
Grilles	1^{er}	$2^{ème}$	$3^{ème}$

Tab. 1 – Comparaisons analytiques des espérances des 3 algorithmes sur différentes familles de graphes (les graphes d'*Avis-Imamura* sont décrits plus en détails dans [1] et [3]).

Références

[1] E. Angel, R. Campigotto, and C. Laforest. Algorithms for the vertex cover problem on large graphs. Technical Report No 1, IBISC – Université d'Evry, 2010.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation.* Springer, 1999.

[3] D. Avis and T. Imamura. A list heuristic for vertex cover. *Oper. Res. Lett.*, 35 :201–204, 2006.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman & Co., New York, 1979.

[1]Notations : n désigne le nombre de sommets du graphe, $d(u)$ désigne le degré (*i.e.* le nombre de voisins) d'un sommet u et Δ désigne le degré maximal du graphe.