3803521 - CANITROT Julien
3800417 - DUBREUIL Jules

SCIENCES SORBONNE UNIVERSITÉ

Rapport Projet - Vertex Cover

UE COMPLEX 2021-2022

Code source : https://github.com/Anidwyd/vertex-cover

Table des matières

Table des matières 1

1 Introduction 2

2 Implémentation des méthodes approchées 2

2.1 Optimalité de l'algorithme glouton 2

2.2 Comparaison des méthodes approchées 3

3 Implémentation des méthodes de branchement 4

3.1 Branchement naïf 4

3.2 Ajout de bornes 4

3.3 Comparaison des méthodes de branchement 6

4 Conclusion 8

1 Introduction

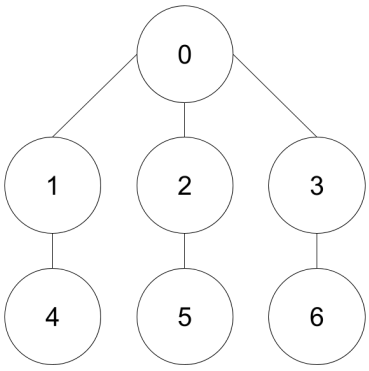
Dans ce projet, nous nous sommes intéressés à l'implémentation d'algorithmes de résolution du problème de **couverture minimum par sommets** (*Vertex Cover* en anglais). Ce problème consiste à déterminer en ensemble minimum de sommets pour couvrir toutes les arêtes d'un graphe.
Nous avons choisi de représenter un graphe par une liste de sommets et une liste d'arêtes.
Les algorithmes peuvent être découpés en 2 catégories:

- les méthodes approchées (couplage et glouton),
- la méthode de branchement (avec ses améliorations).

2 Implémentation des méthodes approchées

2.1 Optimalité de l'algorithme glouton

Voici l'exemple d'un graphe G pour lequel l'algorithme glouton ne renvoie pas la solution optimale.



En effet la solution optimale est $\{1, 2, 3\}$ et est donc de taille 3. L'algorithme glouton va quant à lui sélectionner le sommet de degré maximum (0) auquel il va ajouter une combinaison de 3 sommets entre [1 ou 4], [2 ou 5] et [3 ou 6]. Il renverra donc une solution de taille 4, qui n'est pas optimale.
D'après cet exemple, on peut en déduire que l'algorithme glouton n'est pas r-approché, pour :

$$r < glouton(G) \,/\, OPT(G) = 4/3$$

❒

Pour montrer que cet algorithme n'est pas r-approché pour tout r, on pourrait trouver un algorithme qui construit un graphe H tel que $glouton(H)$ pourrait renvoyer, s'il fait les pires choix à chaque itération, une solution aussi loin que l'on veut de l'optimale.

2.2 Comparaison des méthodes approchées

Pour comparer les temps de calcul de `couplage` et `glouton` nous avons mesuré une taille $Nmax = 1000$ de sommets jusqu'à laquelle les algorithmes tournent rapidement. Nous avons généré une vingtaine de graphes pour chaque $n \in \{Nmax/10, 2Nmax/10, \ldots, Nmax\}$ avec une probabilité $p = 1/\sqrt{n}$ pour obtenir des graphes légers (*Fig.1*).

Pour analyser l'influence de la probabilité p de présence d'une arête, nous l'avons faite varier de 0.1 à 0.9, pour $n = 100$ (*Fig.2*).

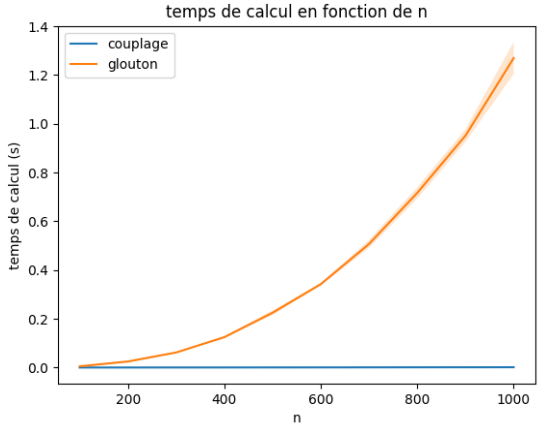


Fig.1

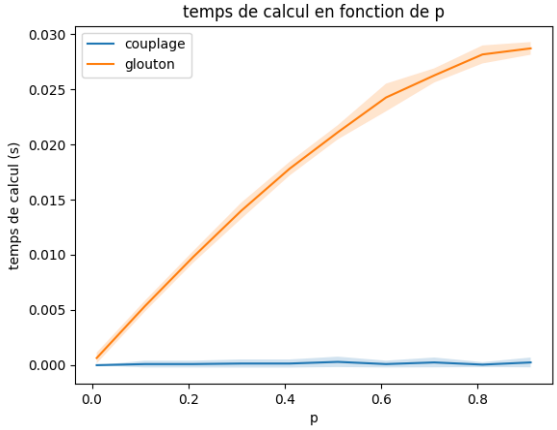


Fig.2

Par lecture graphique, on constate que les variations de temps de calcul (en fonction de n comme de p) de `couplage` sont insignifiantes comparées à celles de `glouton` dont les variations semblent suivre une évolution quadratique en fonction de n et logarithmique en fonction de p.

Ces algorithmes étant des approximations, leur qualité est tout aussi importante que leur vitesse de calcul. C'est pourquoi nous avons comparé leur rapport d'approximation avec la solution exacte, calculée par l'algorithme de branchement amélioré `bb_improved2` que nous verrons plus tard. Nous avons réalisé ces tests pour un $Nmax$ de 100 (*Fig.3*).

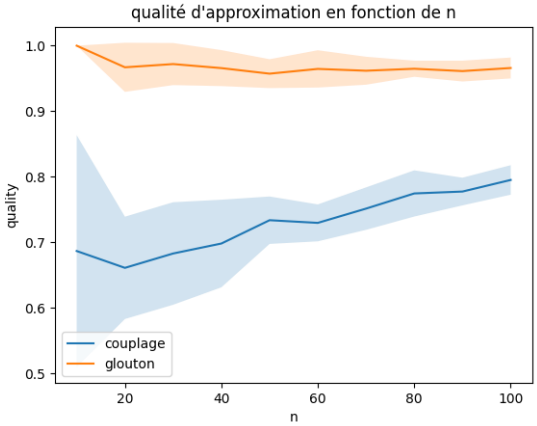


Fig.3

L'algorithme `glouton` obtient une qualité de réponse moyenne de 96% tandis que `couplage` s'améliore en fonction de n, avec une moyenne générale aux alentours de 75%.

Ces 2 algorithmes ont donc leurs avantages et inconvénients. Le couplage est plus rapide et a une garantie de précision, il est 2-approché. L'algorithme glouton quant à lui est très proche de la solution optimale dans les cas généraux mais peut se tromper autant qu'on le veut dans des cas spécifiques. De plus, il est beaucoup plus long à exécuter.

Mais malgré leur précision, ces algorithmes ne sont pas optimaux, d'où l'implémentation d'un algorithme de branchement.

3 Implémentation des méthodes de branchement

3.1 Branchement naïf

Nos algorithmes de branchement utilisent une structure `Node[Gn,Cn,Sn]` permettant de stocker pour chaque nœud de l'arbre, le graphe actuel Gn, la couverture partielle actuelle Cn, et le sommet (ou la liste de sommets) Sn à ajouter dans la couverture.

La version naïve choisit une arête $\{u, v\}$ dans la liste d'arêtes du graphe G et ajoute dans la pile les nœuds $[G, \emptyset, u]$ (u est dans la couverture) et $[G, \emptyset, v]$ (v est dans la couverture).
Tant que la pile n'est pas vide:

- on récupère le dernier nœud ajouté,
- on supprime le (ou les) sommet(s) Sn de Gn et on ajoute Sn à Cn,
- si le nombre d'arêtes de Gn est nul (on est sur une feuille) on compare Cn (solution réalisable) à notre couverture actuelle, que l'on met à jour si besoin,
- sinon, on branche sur une arête de Gn.

Enfin, on retourne la couverture trouvée.

3.2 Ajout de bornes

Montrons la validité des bornes proposées : $b_1 = \lceil \frac{m}{\Delta} \rceil$, $b_2 = |M|$ et $b_3 = \frac{2n-1-\sqrt{(2n-1)^2-8m}}{2}$.

- b_1 : un sommet peut couvrir au plus Δ arêtes. Pour un graphe dont toutes les arêtes sont de degré Δ, il faut $\lceil \frac{m}{\Delta} \rceil$ sommets dans la couverture.

  Ainsi, pour n'importe quel graphe, il faut faut donc une couverture contenant au moins $\lceil \frac{m}{\Delta} \rceil$ sommets, d'où :

  $$|OPT| \geq \lceil \frac{m}{\Delta} \rceil$$

❒

- b_2 : l'algorithme du couplage maximal que nous utilisons renvoie une couverture C composée de sommets et pas d'arêtes.
  On a donc : $|C| = 2 \times |M|$. Or on sait aussi que cet algorithme est 2-approché. On en déduit que :
  $$\frac{2|M|}{|OPT|} \leq 2$$
  D'où :
  $$|OPT| \geq |M|$$
  ❒
- b_3 : Soit G un graphe de n sommets, m arêtes et de couverture C. Soit x le nombre maximal d'arêtes dans C, on a :
  $$|C| \geq x$$
  D'une part, on a $(x-1)x/2$ qui représente le nombre maximal d'arêtes de G dont les deux extrémités font partie de C. D'autre part on a nx qui représente le nombre maximal d'arêtes de G dont une seule des extrémités est dans C.
  On a bien :
  $$m = \frac{(x-1)x}{2} - nx \Leftrightarrow x^2 + (1-2n)x + 2m = 0$$
  En résolvant la racine, on obtient :
  $$x = \frac{2n-1-\sqrt{(2n-1)^2-8m}}{m} = b_3$$
  D'où :
  $$|C| \geq b3$$
  ❒

En prenant $b_{inf} = max(b_1, b_2, b_3)$, on obtient la borne inférieure d'un nœud. On peut aussi définir la borne supérieure b_{sup} comme étant la taille de notre meilleure solution. Par conséquent, si :

$$C_n + b_{inf} \geq b_{sup}$$

On peut se permettre d'élaguer la branche, car le nœud ne peut s'assurer une solution de taille inférieure à la borne supérieure. Sinon on peut déterminer une solution réalisable $Creal$ pour le graphe actuel, et comparer sa taille à b_{inf}.
En effet, si :

$$|Creal| = b_{inf}$$

On peut élaguer et mettre à jour notre couverture C en la remplaçant par $Creal \cup C_n$.

3.3 Comparaison des méthodes de branchement

Afin d'évaluer l'impact des différentes améliorations de l'algorithme de branchement, nous avons réalisé des tests sur l'évolution du temps de calcul en fonction de n.

Nous avons suivi la même procédure qu'au point 2.2 mais en fixant $Nmax$ à 35 pour comparer `branch` avec `branch_bound` (*Fig.4*) puis à 100 pour comparer `bb_improved` avec `bb_improved2` (*Fig.5*).

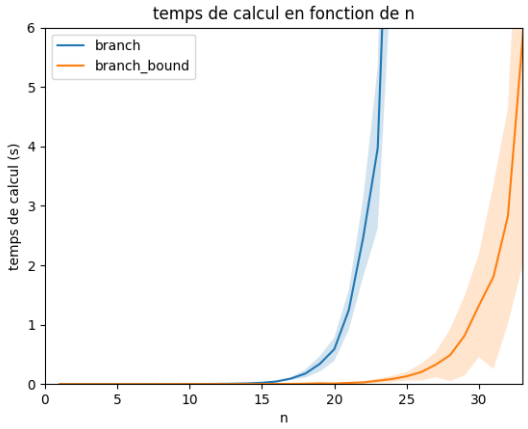


Fig.4

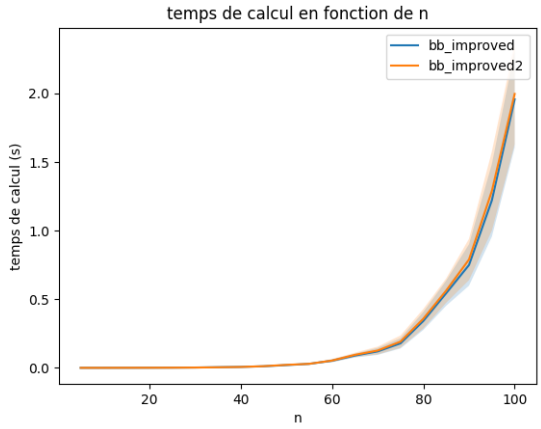


Fig.5

On constate sur la *Fig.4* que les algorithmes `branch` et `branch_bound` dépassent les 6 secondes de calcul pour respectivement 22 et 33 sommets tandis que sur la *Fig.5* les algorithmes améliorés dépassent les 2 secondes pour 100 sommets.
Afin de mieux comparer les algorithmes, les évolutions ayant l'air exponentielles nous avons appliqué une échelle logarithmique sur le temps de calcul (*Fig.6* et *Fig.7*).

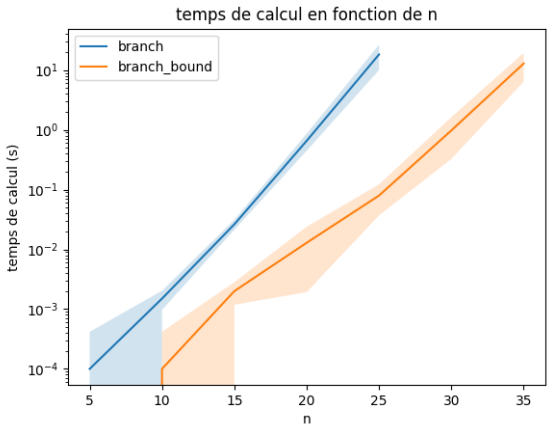


Fig.6

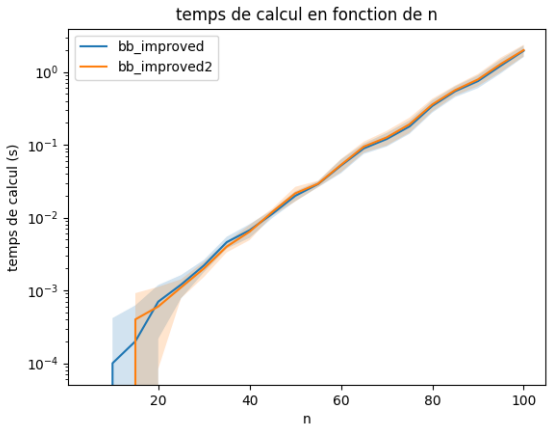


Fig.7

Les nouvelles courbes étant linéaires, on peut affirmer que les évolutions sont exponentielles dans les deux figures.

On a ainsi pu estimer que l'évolution de ces courbe suivait pour :

- les algorithmes `branch` et `branch_bound` semble suivre une suivent une courbe exponentielle similaire (surement peu plus faible pour le `branch_bound` mais les données ne nous permettent pas de trancher) de $10^{0.05x}$, soit le facteur $\alpha = 0.05$,
- les algorithmes améliorés suivent eux aussi une courbe exponentielle mais leur coefficient linéaire est 5 fois plus petit, donc $10^{0.01x}$, $\alpha = 0.01$.

Ainsi les versions améliorées de par leur facteur alpha plus petites permet de donner des résultats pendant beaucoup longtemps ce qui est intéressant.

Pour compléter nos analyses expérimentales nous avons testé l'influence de p sur l'évolution du temps t de calcul en fixant n à 15 (*Fig.8*) puis le nombre de noeuds créés en fonction de n pour $Nmax = 100$ (*Fig.9*).

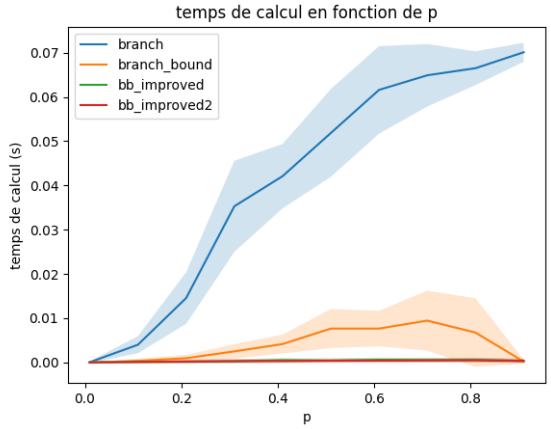


Fig.8

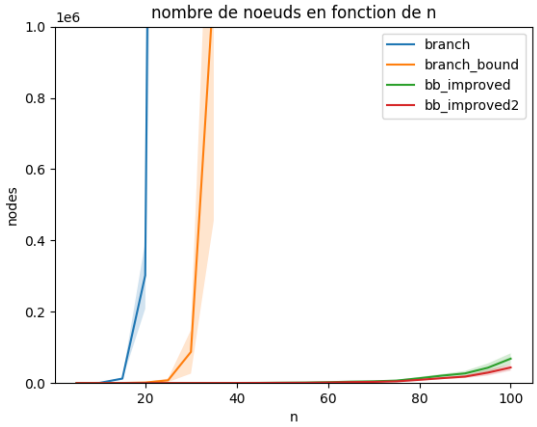


Fig.9

On remarque encore une fois que l'évolution de t ralenti (voire diminue) lorsque p se rapproche de 1. Par lecture graphique de la *Fig.9* on peut noter la forte corrélation entre temps de calcul (*Fig.4)* et nombre de nœuds générés (ce qui semble cohérent).

L'ensemble de ces tests met bien en évidence la qualité des améliorations de `bb_improved` et le peu d'intérêt d'ajouter le test sur les sommets de degré 1.

4 Conclusion

Avec ce projet nous avons pu voir différentes approches pour essayer de résoudre un problème NP-complexe.

On a vu un algorithme approché (`couplage`) peu performant sur l'optimalité de la solution. Cependant il a une garantie sur le résultat approché. Un autre (`glouton`) plus précis en général, mais qui peut se tromper autant que l'on veut pour des instances spécifiques.

Cela nous a amené à créer des algorithmes de branchement qui renvoient la solution optimale mais en temps d'exécution exponentiel. Nous sommes donc passé par une phase importante d'optimisation pour diminuer leur complexité, et leur permettre de résoudre des instances de plus grandes tailles.