Sorbonne Université

UE 4IN900 - « COMPLEX », M1 SFPN

Rapport de projet VERTEX COVER

Calle Viera Andersson

Encadré par
ESCOFFIER Bruno

Table des matières

Introduction

Soit un graphe $G = (V, E)$ non orienté, où V : l'ensemble des sommets du graphe et E : l'ensemble des arêtes de G. Posons dans tout le reste du rapport $|V| = n$ et $|E| = m$.
Une couverture de G est un ensemble $V\prime \subset V$ tel que toute arête $e \in E$ a (au moins) une de ses extrémitées dans V.

L'objectif global de ce projet est donc de trouver une couverture contenant un nombre minimum de sommets.

Exemple de couverture contenant un nombre de sommets minimum.

Le problème étant NP-difficile, le but est d'implémenter différents algorithmes, exacts et approchés, pour résoudre le problème Vertex Cover ainsi que de les tester expérimentalement sur différentes instances.
Les analyses seront faites en **Python** grâce au fichier *VertexCover.py* dans lequel on a implémenté les graphes grâce à la bibliotéque **NetworkX**.

1 Graphes

Dans l'ensemble du rapport lorsqu'il s'agit de réaliser un test, par exemple sur le temps de calcul d'une classe d'instances, nous réaliserons toujours les mêmes étapes.

— Le calcul du N_{max} pour lequel l'algorithme toune rapidement, *i.e* moins de $temps_{max} = 4s$.

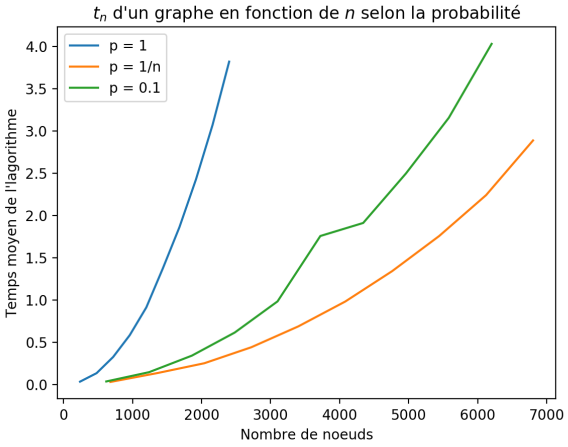
— $\forall n \in \{Nmax/10,\ 2Nmax/10,\ 3Nmax/10, \ldots,\ Nmax\}$. On va générer 10 instances de taille n dont on va calculer le temps moyen t_n.

— On tracera finalement la courbe de t_n en fonction de n avec ,si besoin est, une échelle logarithmique pour pouvoir déterminer la classe de compléxité de l'algorithme.

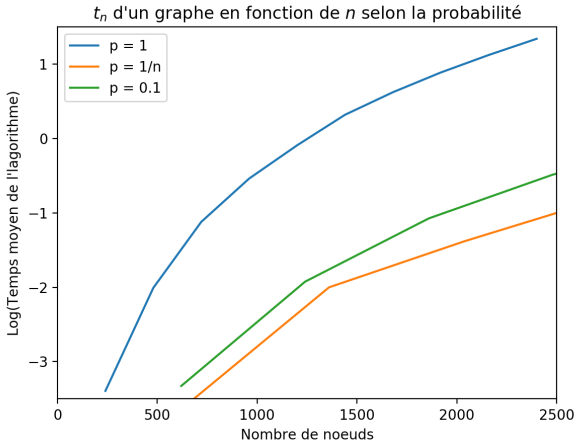
Dans la suite lorsqu'il sera question de réaliser des tests où des probabilités interviennent, sans mention du contraire, on chosira $p_{max} = 1$. En effet un graphe ainsi créé est complet et donc correspond souvent au pire cas de recherche pour les algorithmes implémentés.

Génération d'instances

Ici $N_{max} = 2400$ notre algorithme de création de graphe aléatoire tourne donc rapidement, moins de $4s.$, jusqu'à 2400 sommets.



t_n en fonction de n et de p.

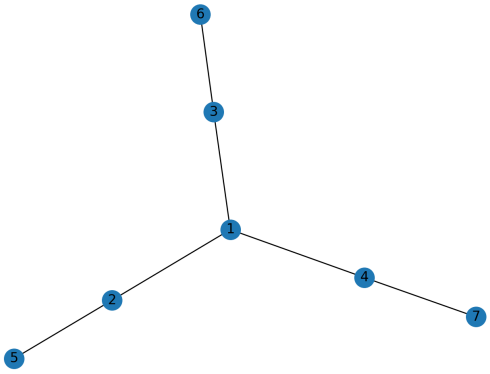


t_n en fonction de $Log(n)$ et de p.

On peut bien remarquer que le temps de calcul est proportionnel à la probabilité, le pire cas est atteint pour $p = 1$. Il est aussi proportionnel au nombre de sommets. Notre intuition était donc bonne ici $p = 1$ correspond effectivement au pire cas.
À première vue on pourrait dire que l'algorithme est exponentiel car la courbe augmente très rapidement. Néanmoins le changement d'échelle nous permet d'observer une tendance logarithmique pour toutes les probabilités. On peut donc affirmer que RandomG est donc polynomiale en fonction du nombre de sommets .

2 Méthodes Approchées

Analyse de algo_glouton :



Graphe pour lequel l'algorithme ne retourne pas une solution optimale.

Pour le graphe ci-dessus on peut vérifier qu'une solution optimale pour $Vertex\text{-}Cover$ est $Opti = \{2,3,4\}$ avec $|Opti| = 3$, tandis que $algo\_glouton$ nous renvoie $Algo = \{1,2,3,4\}$ tel que $|Algo| = 4$. Ainsi comme $|Opti| \geq |Algo|$, l'lagorithme précédent n'est pas optimal.

Montrons alors que pour $\forall r > 0$ il n'est pas non plus r-approché en raisonnant par l'absurde. Soit $r > 0$, supposons que $algo\_glouton$ est r-approché $i.e$ $|Algo] \leq r|Opti|$. Considérons un graphe biparti $G = (U \cup V, E)$ tel que $|U| = n$ et $V = \bigcup_{i=2}^{n} V_i$, où $V_i = \{1; \ldots; \lfloor \frac{n}{i} \rfloor\}$ tel que $\forall v \in V_i : deg(v) = i$.

À létape $j = 1$ $\forall u \in U$ $deg(u) \leq n-1$, or $\exists v \in V$ $deg(v) = n$ donc notre algorithme va choisir ce sommet en question(le seul sommet dans V_n)

Ainsi pour toute étape j $\forall u \in U$ $deg(u) \leq n-j$, or $\exists v \in V$ $deg(v) = n-j+1$ donc notre algorithme va choisir ce sommet en question(le seul sommet dans V_{n-j+1}) Ainsi à chaque étape l'algorithme choisit un sommet de V et termine quand il a choisit tout les sommets de V.
On a donc que $|Algo| = |V|$ alors que $|Opti| = |U| = n$, or :

$$|V| = |\bigcup_{i=2}^{n} V_i| = \sum_{i=2}^{n} |V_i| = \sum_{i=2}^{n} \left\lfloor \frac{n}{i} \right\rfloor$$

$\forall x \in \mathbb{R} : \lfloor x \rfloor \leq x \leq \lfloor x \rfloor + 1 \implies \lfloor \frac{n}{i} \rfloor \geq \frac{n}{i} - 1$

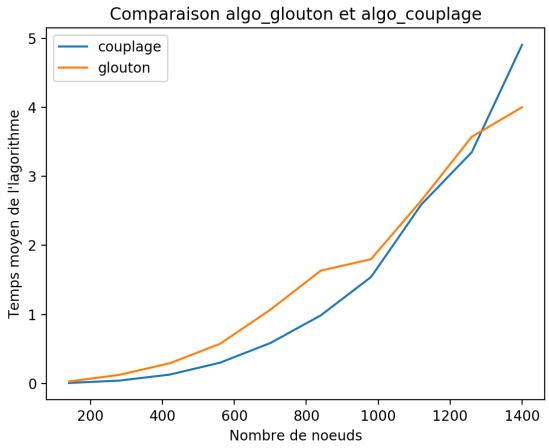
$$\begin{aligned}
&= \sum_{i=2}^{n} \left\lfloor \frac{n}{i} \right\rfloor \\
&\geq \sum_{i=2}^{n} \left(\frac{n}{i} - 1 \right) = n \sum_{i=2}^{n} \left(\frac{1}{i} - \frac{1}{n} \right) \\
&= n \sum_{i=2}^{n} \frac{1}{i} - n \sum_{i=2}^{n} \frac{1}{n} = n \sum_{i=2}^{n} \frac{1}{i} - \frac{n(n-1)}{n} = n \left(\sum_{i=1}^{n} \frac{1}{i} - 1 \right) - n + 1 = n \sum_{i=1}^{n} \frac{1}{i} - 2n + 1 \\
&\geq n \sum_{i=1}^{n} \frac{1}{i} - 2n
\end{aligned}$$

Et donc, $$\frac{1}{n} \left(n \sum_{i=1}^{n} \frac{1}{i} - 2n \right) = \sum_{i=1}^{n} \frac{1}{i} - 2 \leq \frac{|V|}{|U|} \leq r$$

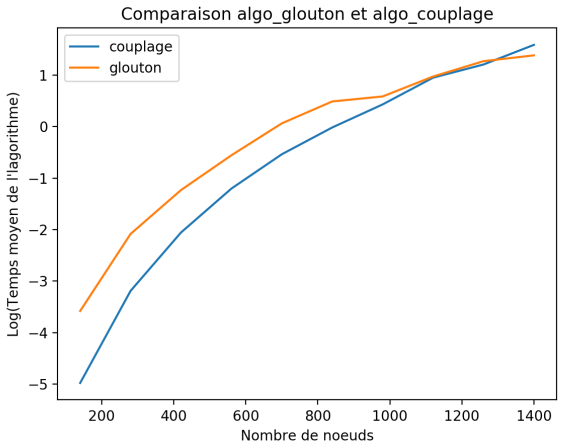
Or $\lim_{n \to \infty} \frac{1}{i} = \infty \implies \frac{|V|}{|U|}$ diverge, contradiction. Donc $|Algo| > r|Opti|$ $i.e$ $algo\_glouton$ n'est pas r-approché. □

Comparaison des algorithmes :

Temps de calcul :



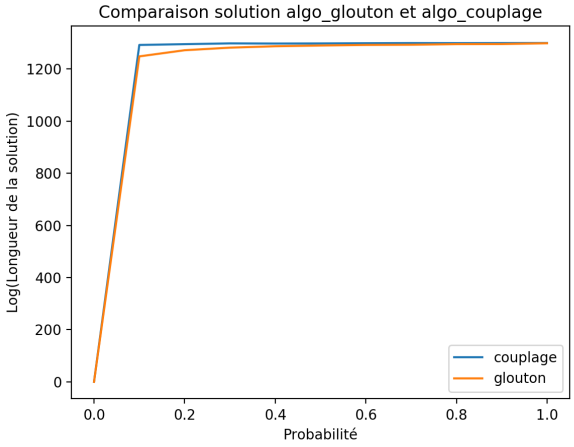
Temps de calcul couplage v. glouton



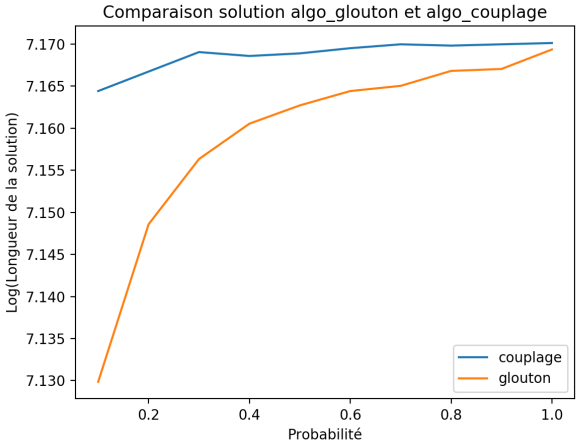
Log(Temps de calcul) couplage v. glouton

On peut donc voir que pour $n \in [\![0, 1300]\!]$ *algo_couplage* est plus rapide mais à partir de n>1300 *algo_glouton* commence à être plus efficace. Grâce à l'échelle logarithmique on observe que les deux algorithmes semblent suivre une tendance polynomiale. Globalement *algo_couplage* est plus efficace niveau temps de calcul.

Qualité des solutions :



Qualité des solutions couplage v glouton



Log Qualité des solutions couplage v glouton

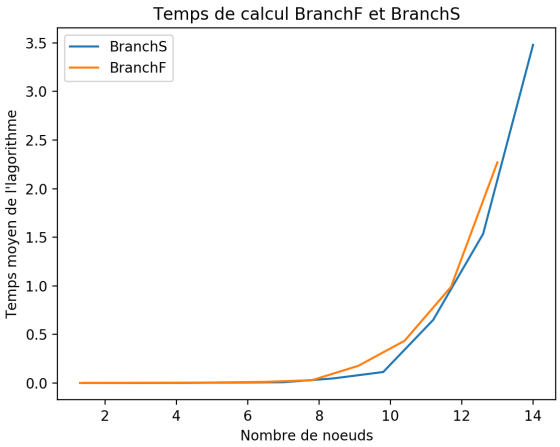
Même si *algo_couplage* est généralement plus rapide, *algo_glouton* nous fournit des solutions moins longues et donc qui correspondent plus vraissemblablement à une couverture minimale d'un Graphe.

On peut remarquer néanmoins que *algo_couplage* semble etre plus constant que *algo_glouton*. Par rapport à la qualité de solution c'est *algo_glouton* qui est plus efficace.

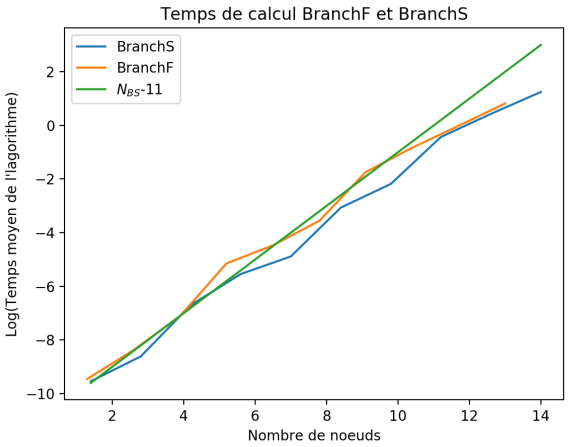
3 Séparation et évaluation

Branchement :

Temps de calcul de la méthode en fonction de n :



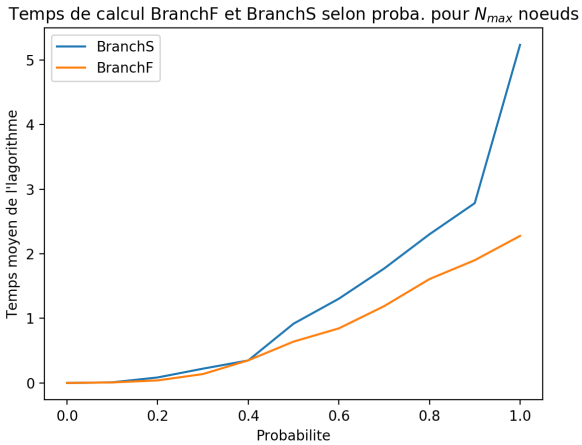
Temps de calcul Branchement



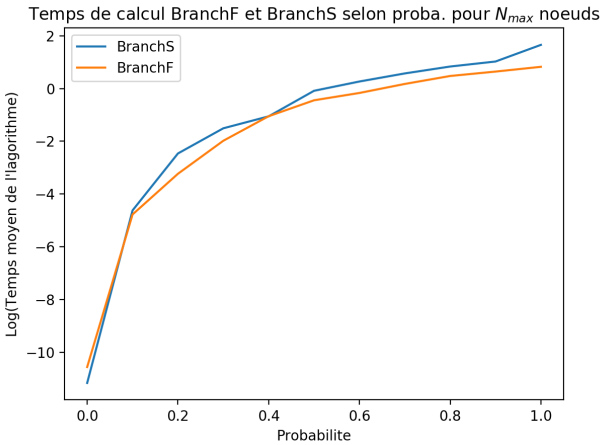
Log Temps de Branchement

Ci-dessus on peut observer le temps de calcul de deux implémentations de l'algorithme de branchage, *BranchF* s'inspire de la version récursive tandis que *BranchS* est moins longue.
On observe que malgré le fait que BranchS nous permette de traiter les graphes avec un sommet de plus, les deux implémentations sont assez proches niveau temps de calcul avec un léger avantage pour *BranchS*. Si on affiche les résultats en échelle logarithmique ainsi que le nombre de sommets en fonction du Log($nb_{sommets}$) - 11, qui est linéaire, on observe que nos courbes ont presque la même allure. Cela veut donc dire que ces algorithmes sont exponentiels en fonction du nombre de sommets.

Temps de calcul de la méthode en fonction de la probabilité p :



Temps de calcul Branchement



Log Temps de Branchement

À première vue la deuxiéme implémentation semble beaucoup plus touchée par l'augmentation de la densité du graphe. Une fois l'échelle logarithmique affichée on se rend compte que finalement ces deux algorithmes qui semblent être polynomiales en fonction de la probabilité d'une arête ne sont pas si éloignés que ça avec toujours un avantage pour *BranchF*.

Ajouts de bornes :

Choix des bornes Considérons : G, un graphe non orienté comme présenté en introduction avec les mêmes notations, M un couplage de G, C une couverture de G, $\Delta = \underset{v\in V}{max}\ deg(v)$ ainsi que :

$$b_1 = \lceil \frac{m}{\Delta} \rceil \qquad b_2 = |M| \qquad b_3 = \frac{2n - 1 - \sqrt{(2n-1)^2 - 8m}}{2}$$

alors : $|C| \geq max\{b_1, b_2, b_3\}$
Montrons que ces bornes sont valides :

— Supposons que $b_1 = max\{b_1, b_2, b_3\}$:
• Si $\Delta | m$ alors $\lceil \frac{m}{\Delta} \rceil = \frac{m}{\Delta}$, de plus on sait que : $\sum_{v\in V} deg(v) = 2m$, or $C \cup (V \setminus C) = V$ donc $\sum_{v\in V} deg(v) = \sum_{v\in C} deg(v) + \sum_{v\in V\setminus C} deg(v) = 2m$, or $\sum_{v\in V\setminus C} deg(v) \leq m$ donc
$\sum_{v\in C} \Delta \geq \sum_{v\in C} deg(v) \geq m \iff |C|\Delta \geq \sum_{v\in C} deg(v) \geq m \iff |C| \geq \frac{m}{\Delta}$

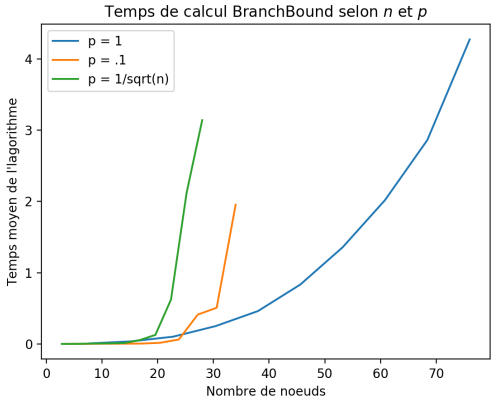
• Sinon comme $\Delta \leq m$ alors $\lceil \frac{m}{\Delta} \rceil \leq m$. On conclut de la même façon.

— Supposons que $b_2 = max\{b_1, b_2, b_3\}$:
Posons $M = \{m_1; m_2; m_3; \ldots\}$ alors $\forall m_i \in M$, posons $m_i = (m_i^1; m_i^2)$. Par définition de $M, \forall i \neq j : (m_i^1; m_i^2) \bigcap (m_j^1; m_j^2) = \varnothing$, de plus $C \subset V$ est definit tel que $\forall (u; v) \in Eu \in Couv \in C$. On peut donc "construire" C tel que $\forall m_i$, $m_i^1 \in C$ ou $m_i^2 \in C$, si on n'a pas couvert toutes les arêtes on rajoute les sommets nécessaires et donc par construction $|C| \geq |M|$

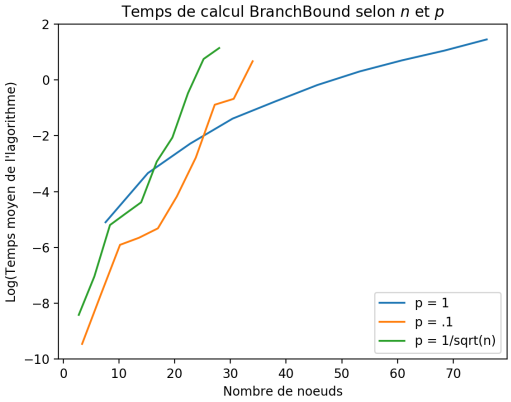
— Supposons que $b_3 = max\{b_1, b_2, b_3\}$:
On va raisonner sur le nombre maximal d'arêtes de G lorsque $|C| = n_0$

□

Test de la méthode avec bornes :

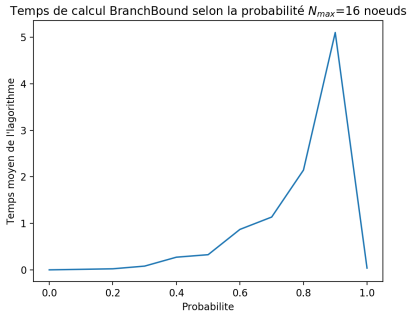


Temps de calcul BranchementBound



Log Temps de BranchementBound

Ci-dessus nous avons afficher l'évolution du temps de *BranchBound* selon le nombre de sommets pour trois cas de probabilités pour une arête. On peut remarquer qu'ici le pire cas semble être $p = \frac{1}{n}$ pour lequel on ne peut traiter au maximum un graphe de $N_{max} = 28$ sommets dans un temps raisonnable. Tandis que l'algorithme semble être polynomiale pour $p = 1$, il semble être exponentiel pour les deux autres cas.

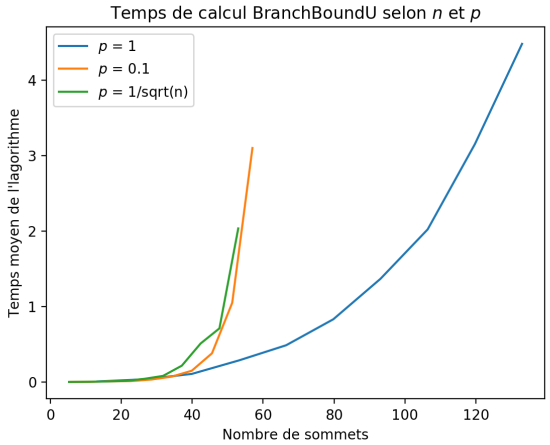


Temps de calcul BranchBound selon p

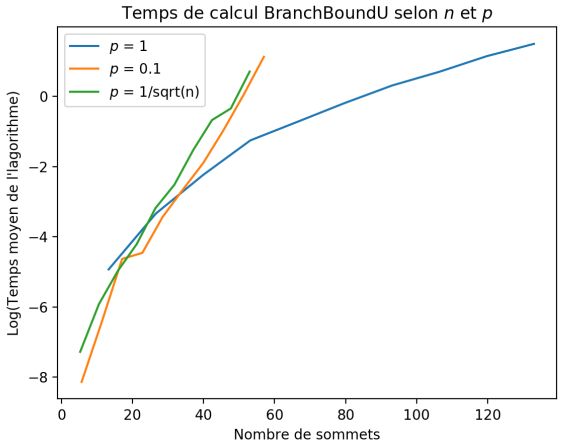
L'affichage du temps de calcul en fonction de la probabilité nous permet d'observer une augmentation nette du temps jusqu'à $p = 0.9$, pour $p = 1$ on retombe à un temps de calcul proche de celui de $p < 0.4$. Cela peut être dû au fait que dans un graphe complet une couverture minimale consiste à prendre n-1 sommets, donc on a trouvé la solution optimale au bout de la première branche de l'arbre.

Amélioration du branchement :

Méthode 1 :



Temps de calcul BranchementBoundU

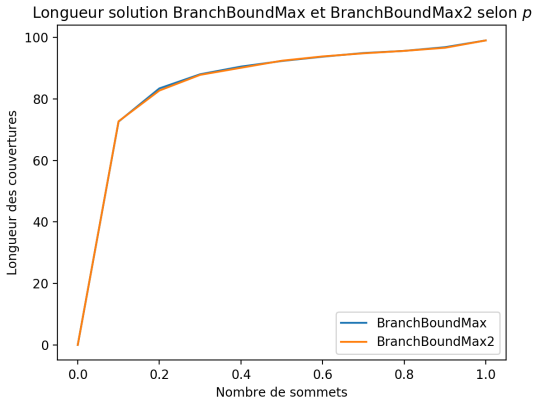


Log Temps de BranchementBoundU

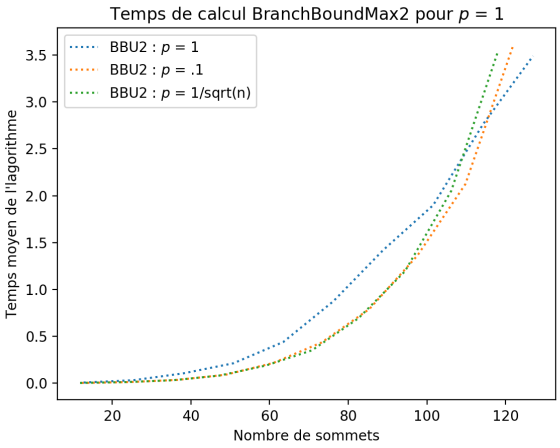
Nous pouvons remarquer que comme pour le premier algorithme qui utilise la borne inf, le pire cas semble être atteint pour $p = \frac{1}{n}$ pour lequel $N_{max} = 53$. Même si on a réussi à doubler notre nombre de sommets traités, la tendance globale de l'algorithme reste la même que le précédent.

Méthode 2 :

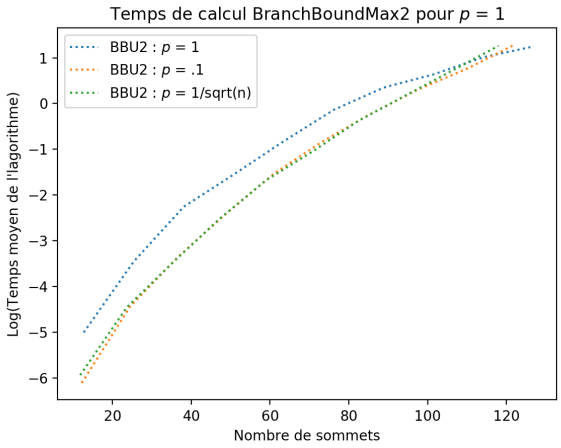
J'ai réalisé deux implémentations de cet algorithme qui différent sur la comparaison avec la borne inf. Les deux semblent retourner des solutions correctes *i.e* des couvertures minimales. Un test sur la longueur des solutions nous permet de voir qu'on obtient généralement la même longueur pour les deux algorithmes même si le deuxième semble être légérement plus court. Il est aussi à noter que le premier algorithme nous permet de traiter rapidement des graphes ayant jusqu'à 135 sommets tandis que le premier s'arrête à 120 sommets. Néanmoins on réalisera par la suite les tests sur BranchBoundMax2.



Comparaison solution des deux algos



Temps de calcul BranchementBoundMax2

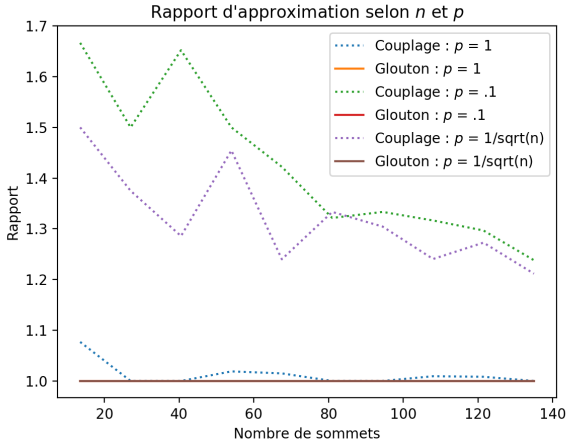


Log Temps de BranchementBoundMax3

On peut remarquer que pour cet algorithme le temps de calcul est plus important pour $p = 1$ jusqu'à 108 sommets, à partir de là c'est $p = \frac{1}{n}$ qui représente le pire cas. Néanmoins touts ces algorithmes ont une tendance polynomiale en fonction du nombre de sommets du Graphe traité.

Qualité des algorithmes utilisés :

Évaluation du rapport d'approximation en fonction de n :
Pour réaliser la calcul des rapport d'approximation nous avons génerer des graphes aléatoires de n sommets $\forall n \in \{Nmax/10,\ 2Nmax/10,\ 3Nmax/10, \ldots,\ Nmax\}$ où $Nmax = 135$. Puis nous avons calculer une couverture minimale à l'aide de $algo\_couplage$, $algo\_glouton$ et $BranchBoundMax2$. Nous avons ensuite calculer les rapports $\frac{algo\_couplage}{BranchBoundMax2}$ et $\frac{algo\_glouton}{BranchBoundMax2}$ réspectifs.



Rapport d'approximation ρ selon p

On remarque bien que, à chaque fois, le rapport d'approximation pour $algo\_glouton$ est non seulement en dessous de celui de $algo\_couplage$ mais il est aussi et surtout égale à 1. On obtient donc à chaque fois la couverture minimale. Ainsi malgré un temps de calcul un peu plus élevé $algo\_glouton$ nous fournit néanmoins de meilleures solutions.

Conclusion

De tout les algorithmes implémentés, *algo_glouton* s'avére être à la fois le plus simple mais aussi le plus efficace dans le problème de la couverture minimale, ce qui parrait assez contre-intuitif à première vue. Si on privilégie la rapidité de calcul sur un graphe avec peu de sommets mais aussi très dense un algorithme de branchement est plus convenable.
En revanche si notre graphe à un nombre plus important de sommets l'un des deux premier algorithmes est à privilégier surtout avec les dernières heuristiques en effet en travaillant sur le dernier algorithme nous avons un nombre de noeuds traitès bien inférieur aux autres.

On pourrait imaginer un algorithme hybride qui en fonction du nombre de sommets et de la fréquence d'apparition d'une arête choisirait quel algorithme est le plus adapté pour minimiser le temps de calcul d'une couverture minimale.