Couverture de Graphe, VERTEX COVER.
ABITBOL Yossef, 3804139,
SERRAF Dan, 3971120.

SORBONNE UNIVERSITÉ

RAPPORT PROJET COMPLEX

COUVERTURE DE GRAPHE

Langage choisi: **python \ Jupyter notebook.**

I. INTRODUCTION :

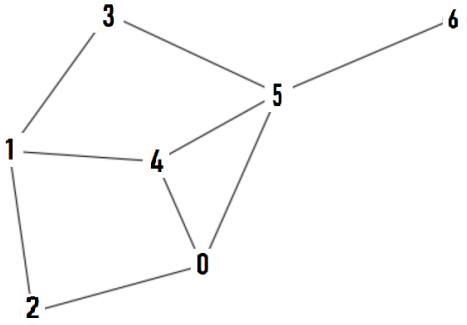
En partant d'un graphe **G = (V,E)** non orienté, avec **V** l'ensemble des n sommets et **E** l'ensembles des m arêtes de G, le but du projet va être de trouver une couverture **contenant un nombre minimum de sommets de G.** C'est-à-dire un ensemble V' inclus dans V tel que toute arête e appartenant a E a au moins une de ses extrémités dans V.

Pour cela, on va implémenter différents algorithmes, exacts et approchés que nous allons tester expérimentalement sur des instances générées aléatoirement afin de les comparer sur leur **temps de calculs** et leur **qualité de réponse**.

II. Graphes :

Pour réaliser les différents algorithmes, nous avons choisi d'utiliser le langage 'python' et de représenter les graphes par un **dictionnaire** dont les clés seront les sommets et les valeurs une liste des arêtes adjacentes :

Graphe = {**0**: [5, 4, 2], **1**: [2, 3, 4], **2**: [1, 0], **3**: [1, 5], **4**: [0, 1, 5], **5**: [0, 3, 4, 6], **6**: [5]}

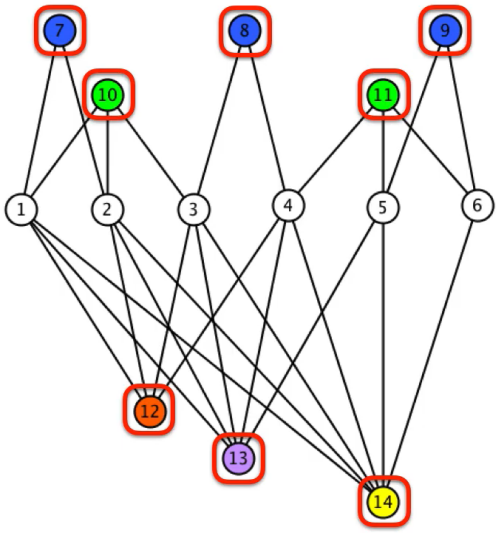


Toutes les fonctions demandées ont été implémentées et testées sur des instances simples avant de les comparer sur des instances générées aléatoirement.

III. MÉTHODES APPROCHÉES :

1) L'algorithme glouton implémenté n'est pas optimal.

CONTRE EXEMPLE :



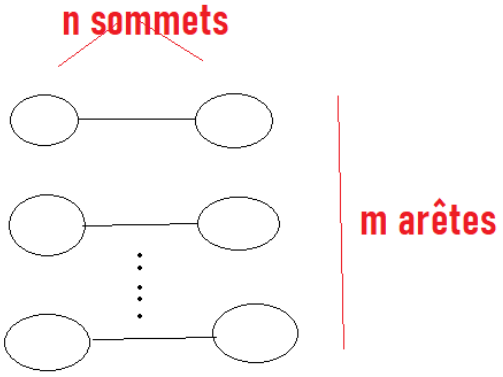
Avec ce graphe, l'algo glouton va prendre les **8 sommets colorés**, en ordre, 14 (6 arêtes incidentes),13,12,11... alors qu'il suffit de prendre les **6 sommets** (1,2,...,6). Ceci met bien en évidence que cet algo n'est pas toujours optimal.

- algoGlouton renvoie C = {14,13,12,11,10,9,8,7}, |C| = 8
- OPT = {1,2,3,4,5,6} , |OPT| = 6
- Ici, **|C|=** $\frac{4}{3}$**OPT**
- Quel que soit ε>0 , algoGlouton n'est pas **(** $\frac{4}{3}$**-** ε **)-approché** $\forall \varepsilon > 0$

2) AlgoCouplage vs AlgoGlouton

Complexité théorique :

Soit un graphe G avec **n** sommets et **m** arêtes.

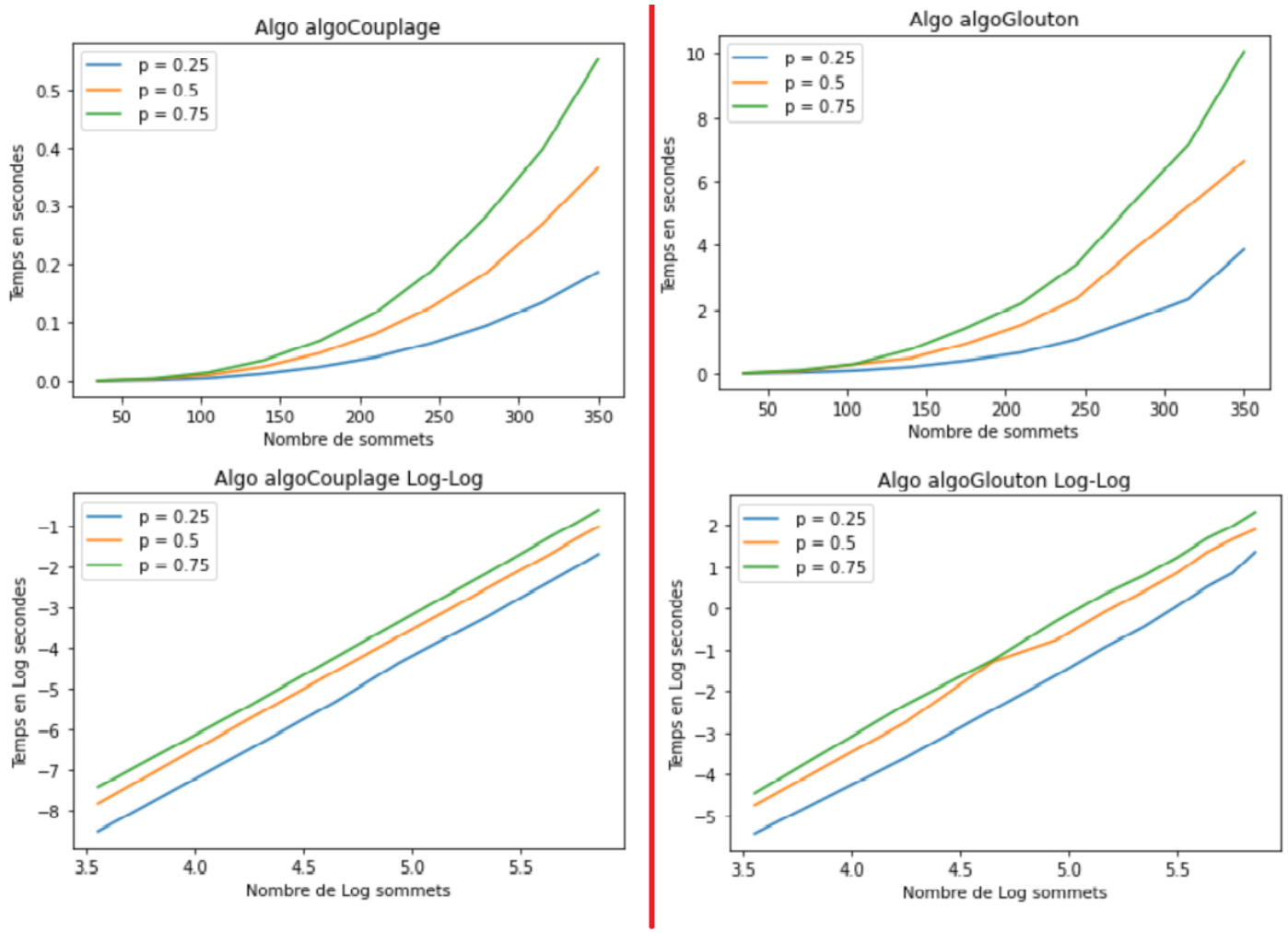


Couverture de Graphe, VERTEX COVER.
ABITBOL Yossef, 3804139,
SERRAF Dan, 3971120.

Algo Couplage : Dans le pire des cas, on a m arêtes avec aucune extrémité en commun. On va donc parcourir tous les sommets en **O(n)** et toutes les arêtes en **O(2*m)** (2*m car on implémente dans le dictionnaire les arêtes dans les 2 sens, {1 : [2] , 2 : [1] }), de plus dans ce cas m = 2/n donc **O(2*m) = O(n)** . Pour chaque arête parcouru on va vérifier si aucun des 2 sommets n'appartient pas déjà à la liste de couverture, dans notre pire cas on va se retrouver avec une couverture contenant tous les sommets donc la vérification se fera en **O(2*n) :** La complexité de l'algo est donc en : **O**(n^3)

Algo Glouton : Dans le pire cas, on a m arêtes avec aucune extrémité en commun, et chaque sommet de degré 1. On va donc parcourir m arêtes en **O(m)** avec **m=n/2**. A chaque itération, on va chercher le degré maximum, pour cela on doit parcourir une fois le dictionnaire pour trouver le nombre de degrés de chaque sommet et une deuxième fois pour les comparer, donc en **O(**n^2**)**: On a donc une complexité en : **O**(n^3)

Comparaison : On fait varier **n [35:350:35]** ainsi que **p [0.25:0.75:0.25]**. Nous avons exécuté chaque cas 10 fois pour ensuite faire la moyenne et obtenir un résultat plus cohérent.



Couverture de Graphe, VERTEX COVER.
ABITBOL Yossef, 3804139,
SERRAF Dan, 3971120.

INTERPRÉTATION :

Comme vous pouvez le voir, on a 4 graphiques, chaque graphique correspond au temps d'exécution de l'algorithme (AlgoCouplage et AlgoGlouton) en fonction du nombre de sommets n. De plus, chaque graphique est composé de 3 courbes qui correspondent aux courbes obtenues lorsqu'on fait varier le paramètre p (0.25,0.50,0.75).

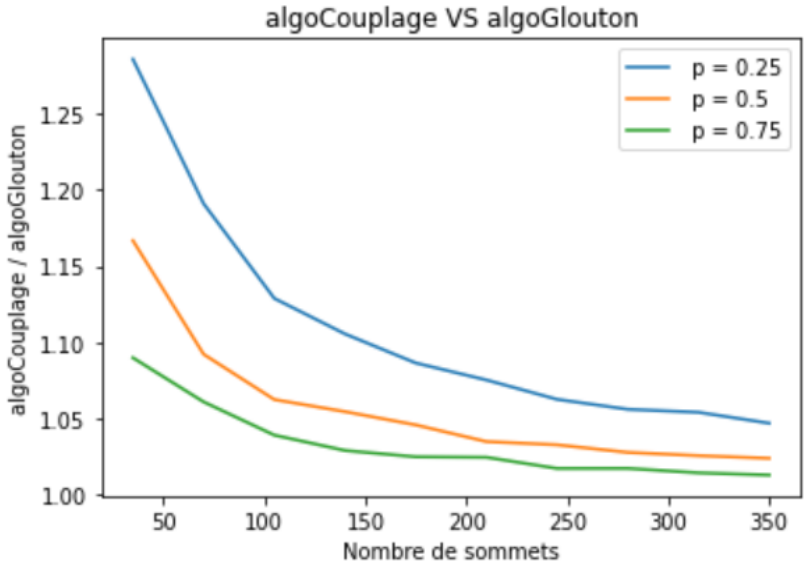
Les 2 graphiques du haut, Algo Glouton a droite et Algo couplage à gauche, mettent en évidence l'algorithme **le plus rapide** qui est celui de l'algo Couplage. Pour une valeur p = 0.5 le temps d'exécution de l'algo couplage est compris entre [0.05 avec n = 50 : 0.4 avec n = 350] contre [1 : 6] pour le glouton. Plus le nombre de sommets augmente, plus le temps est long et plus p augmente, plus le temps est long. **Le nombre de sommets et la valeur de p ont donc un impact direct sur le temps d'exécution.** Nous ne pouvons rien conclure de plus a partir de ces 2 graphiques, c'est pour cela que nous avons tracé les 2 graphiques du bas qui correspondent aux courbes log / log.

En effet, le tracé des 2 graphiques en log nous permet d'avoir une idée sur la **complexité de l'algorithme**, dans les 2 cas on obtient une fonction affine, ce qui signifie que la complexité de part et d'autre est polynomiale. De plus, si nous calculons le coefficient directeur de chaque graphique :

- Couplage : [2.9688, 2.9619, 2.9653] avec p = [0.25 , 0.5 , 0.75]
- Glouton : [2.9450, 2.8827, 2.9391] avec p = [0.25 , 0.5 , 0.75]

Le coefficient directeur est donc à peu près le même pour les 2 algos d'une valeur de 3, on a **une complexité polynomiale de degré 3** pour les 2 algos.

⇒ L'expérimentation est donc en accord avec la théorie. **Les deux algos sont donc bien de complexités polynomiales de degré 3.**



Maintenant, on compare la qualité du résultat, cette qualité est évaluée en comparant la taille de la couverture renvoyée. On a donc décidé de faire un **graphique sur le rapport couplage / glouton.** On constate que **l'algo Glouton renvoie une couverture avec moins de sommets que**

l'algo Couplage. Plus p augmente, plus le rapport diminue et plus le nombre de sommet augmente plus le rapport diminue. Cela signifie, que pour un n très élevé et un grand p, le rapport est très proche de 1 mais pour un petit n et un petit p, la différence se fait observer.

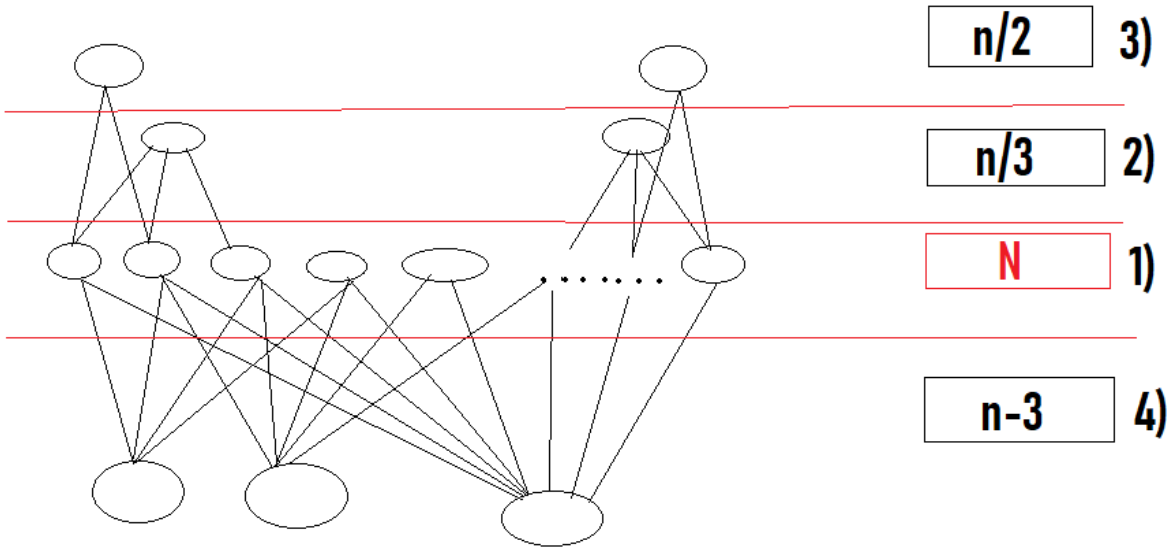
L'algo glouton est donc meilleur au niveau qualité.

Après avoir comparé les 2 algos en fonction de n et de p, on conclut que **l'algo couplage à un meilleur temps de calcul** mais une **qualité de réponse moins optimal que l'algo glouton**.

3) Glouton r-approché ?

On va généraliser le cas du contre exemple dans 1) c'est a dire qu'on va prendre :

1. **n sommets central (1,2,..,n)**, multiple de 3 et de 2, qui va correspondre a la solution OPT.
2. $\frac{n}{3}$ **sommets**, chaque sommet aura 3 arêtes vers les sommets principaux, exemple : sommet a :(1,2,3), b:(4,5,6)... ,(n-2,n-1,n).
3. $\frac{n}{2}$**sommets**, chaque sommet aura 2 arêtes vers les sommets principaux, exemple: sommet a':(1,2), …, (n-1,n).
4. $(n-3)$ **sommets**, le premier sommet à 4 arêtes vers les arêtes central a'': (1,2,3,4) , le deuxième 5 arêtes et le dernier sur les n arêtes.



L'algo glouton va donc renvoyer tous les sommets de 4) ensuite les sommets de 2) et enfin ceux de 3) ce qui donne une couverture :

$$|C| = (n-3) + \frac{n}{3} + \frac{n}{2} \Rightarrow |C| = \frac{11*n-18}{6}$$

or une solution optimal est de prendre les n : $OPT = n$

en faisant le rapport, on obtient : $\frac{|C|}{OPT} = \frac{11}{6} - \frac{18}{6n}$ **avec** $n \to \infty$ **on a** $\frac{11}{6}$ **:**

Cela montre que AlgoGlouton n'est pas $\frac{11}{6} - \varepsilon$-approché quel que soit $r < \frac{11}{6}$ et $\varepsilon > 0$.

IV. SÉPARATION ET ÉVALUATION:

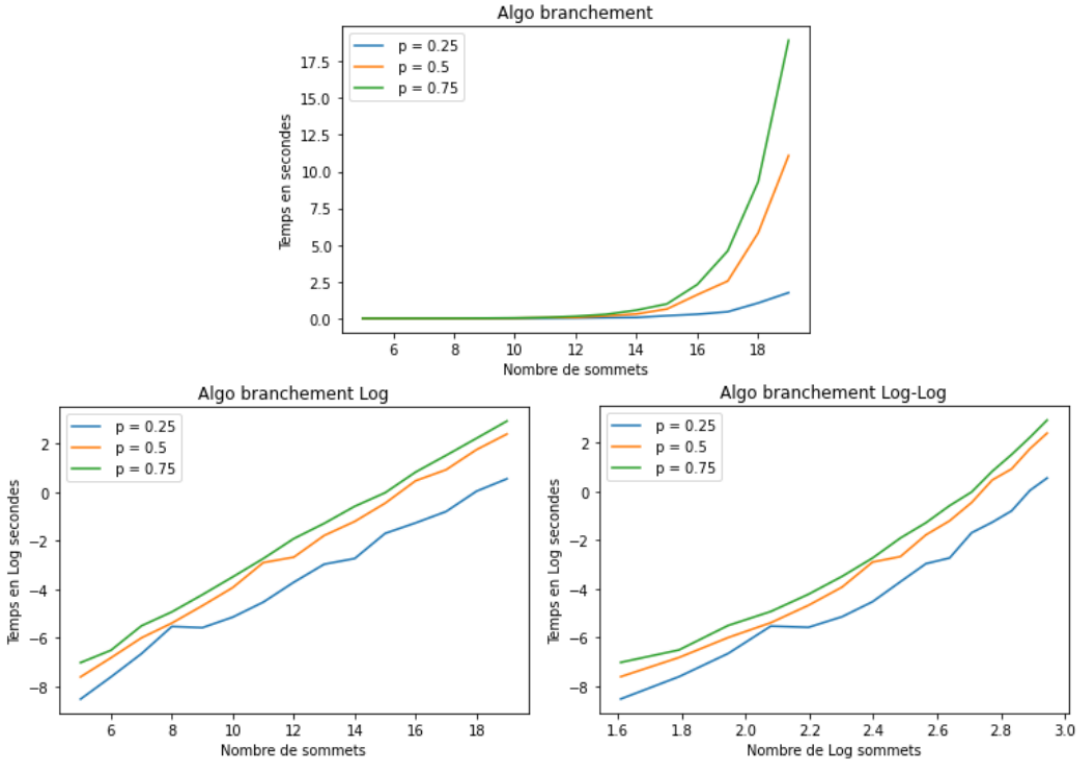
1) Branchement Simple :

Le branchement considéré ici est le suivant : prendre une arête e = {u, v}, et considérer deux cas : la première branche consiste à mettre u dans la couverture,et symétriquement la deuxième consiste à mettre v est dans la couverture.

Complexité théorique:

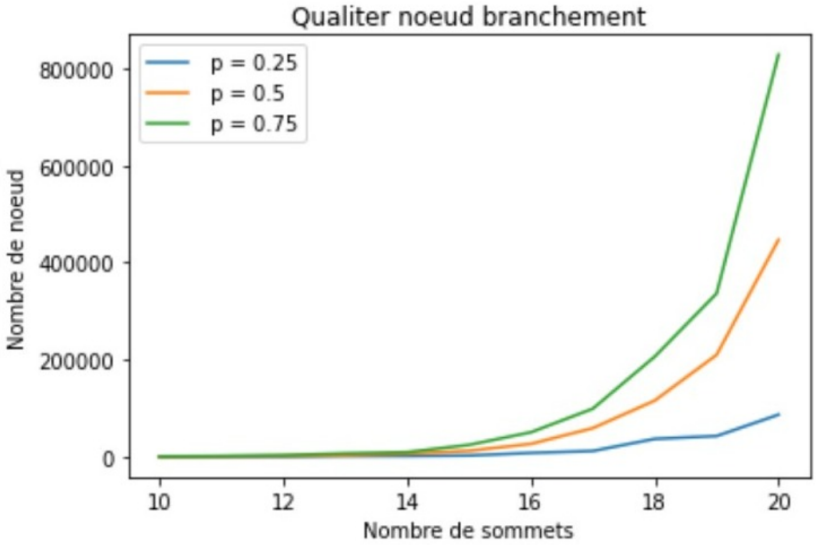
Considérons le pire cas, avoir un **graphe complet** de n sommets et de $m = \frac{n*(n-1)}{2}$ arêtes. Notre algorithme initialise tout d'abord la couverture a vide, le nombre de nœud a 0, u et v et enfin la pile, cela se fait en **O(1).** Ensuite, on itére tant que la pile n'est pas vide, sauf que pour un graphe complet la meilleure couverture est de taille **n-1**, l'algo va donc devoir générer un arbre complet, soit $2^n - 2$ noeuds générés, $2^n - 2$ itérations. La complexité de notre algorithme est donc **exponentielle**.

Nous avons ici choisis : **n[5:25:1]** et **p [0.25:0.75:0.25]** :



Sur la première courbe on peut voir que pour n = 16 on a un temps de calcul moyen de **2 sec** mais lorsqu'on augmente n de 4 on obtient un temps de calcul de **18sec** pour un p = 0.75, **un temps très élevé** car on a vu plus haut que pour n=350 l'algo couplage renvoyé un résultat certe moins optimale, mais au bout de 0.5sec et 10sec pour l'algo Glouton. Plus on augmente n et p, plus le temps augmente de **manière exponentielle**. En effet, comme on peut le constater sur la deuxième courbe, le temps de calcul en log en fonction de n affiche une **droite affine** de coefficient directeur = [0.648, 0.714, 0.711] =**0.7.**

La dernière figure confirme notre hypothèse par son allure qui correspond à une courbe exponentielle.



Cette figure correspond aux nombres de nœuds en fonction de n et de p. Elle confirme bien le fait que cette algo **génère beaucoup de noeuds de manière exponentielle** par exemple pour n = 16 et p = 0.75 on a généré moins de 100 000 noeuds alors que pour n= 20 on a plus de **800 000 noeuds, P** a aussi un impact car plus p augmente plus le nombre de noeuds augmente considérablement.

⇒ Cette algorithme est donc de **complexité exponentielle.** Le problème de cet algorithme est qu'il calcul chaque branche sans se préoccuper de savoir si le parcours de ce nœud est utile et donc **génère beaucoup de nœuds.** Cela nous empêche aussi de tester cet algorithme avec un n très élevé. Comme dans notre pire cas, il génère $2^n - 2$ nœuds alors qu'il obtient la solution optimale avec la première branche.
C'est pour cela qu'on va rajouter des bornes.

2) Ajout de bornes :

Soit G un graphe, M un couplage de G et C une couverture de G. Alors: **|C| ≥ max{b1, b2, b3}**
avec $b1 = [\frac{m}{\Delta}]$ (où Δ est le degré maximum des sommets du graphe), $b2 = |M|$, $b3 = \frac{2*n-1-\sqrt{(2*n-1)^2-8*m}}{2}$.

VALIDITÉ :

Pour $b1 = [\frac{m}{\Delta}]$ il ya trois cas :

- $\Delta > m$: Impossible,
- $\Delta = m$: Toutes les arêtes sont issus du même sommet donc $|C| = 1 \geq \frac{\Delta}{m} = 1$

- $\Delta < m$: Au maximum un sommet a $m - 1$arêtes incidentes et donc on aura besoin d'un deuxième sommet pour couvrir toutes les arêtes, donc : $|C| = 2 \geq [\frac{m}{m-1}]$

Pour $b2 = |M|$:

- On a vu que l'algo Couplage renvoie un couplage maximal, donc on peut prendre le résultat comme un borne inférieur de |C|.

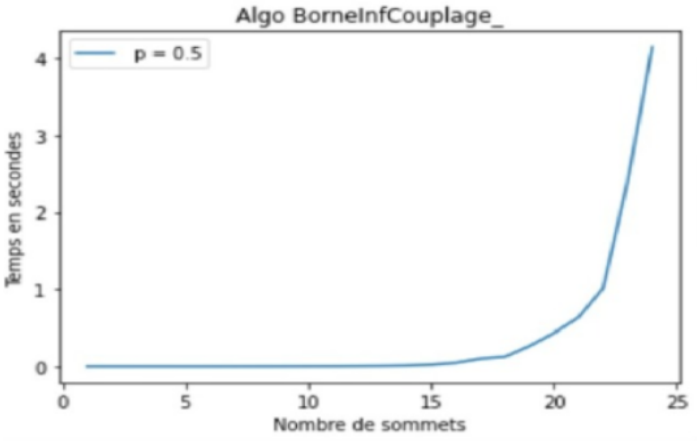
Pour $b3 = \frac{2*n-1-\sqrt{(2*n-1)^2-8*m}}{2}$:

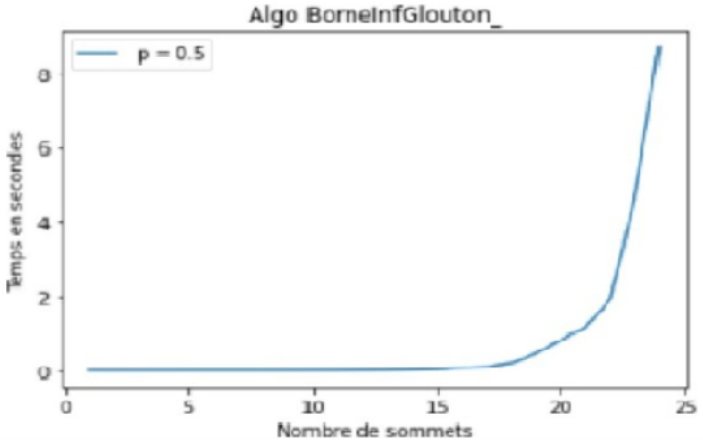
- On prend le cas d'un graphe complet, chaque sommet a une arête incidente vers tous les autres sommets, ce qui se traduit par le nombre d'arêtes **n** et le nombre de **sommet** $m = \frac{n*(n-1)}{2}$**,** on remplace dans la formule :

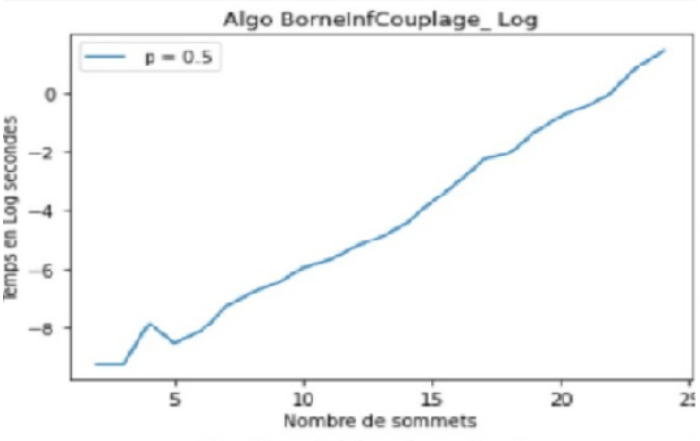
$$b3 = \frac{2*n-1-\sqrt{(2*n-1)^2-8*\frac{n*(n-1)}{2}}}{2} \Rightarrow \frac{2*n-1-\sqrt{(2*n-1)^2-4n^2+4n}}{2} \Rightarrow \frac{2*n-1-\sqrt{4n^2-4n+1-4n^2+4n}}{2}$$

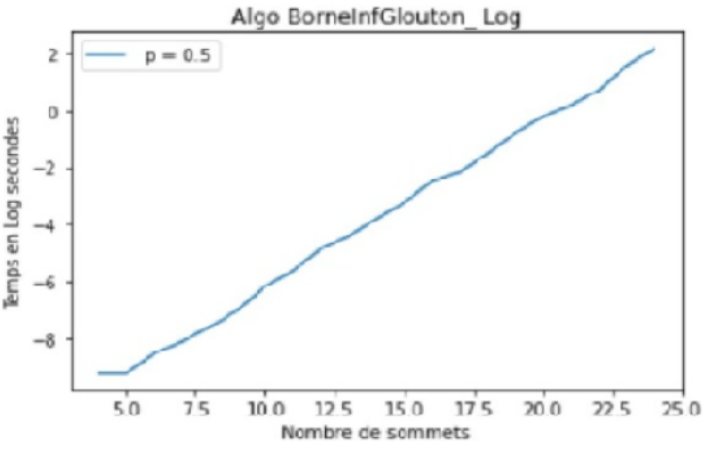
$$\Rightarrow \frac{2*n-1-\sqrt{1}}{2} \Rightarrow \frac{2*n-2}{2} \Rightarrow n - 1 \leq |C| \text{ vraie pour un graphe complet.}$$

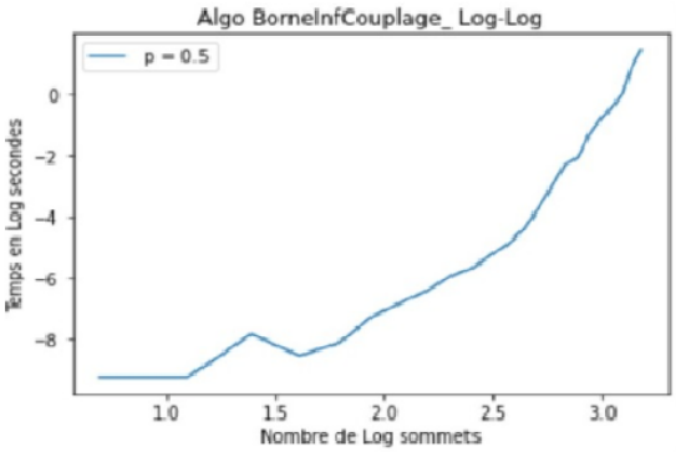
On va modifier l'algorithme vu précédemment, celui du branchement simple en calculant, **en chaque nœud**, une solution réalisable obtenue soit avec l'algo de Couplage soit avec l'algo Glouton (qui renvoyait une solution plus optimal que l'algo Couplage), et le calcul d'une borne inf basé sur la relation précédente pour savoir si il est utile de parcourir ce noeud ou pas.

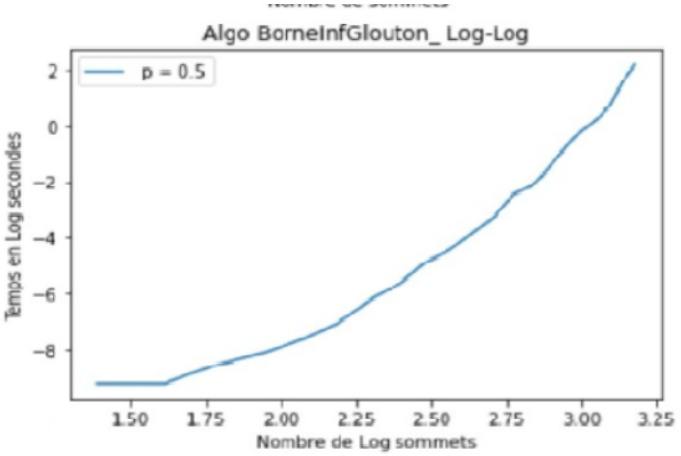




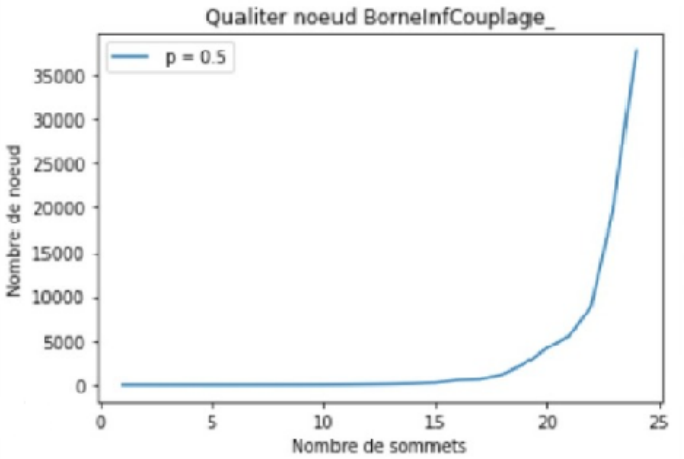


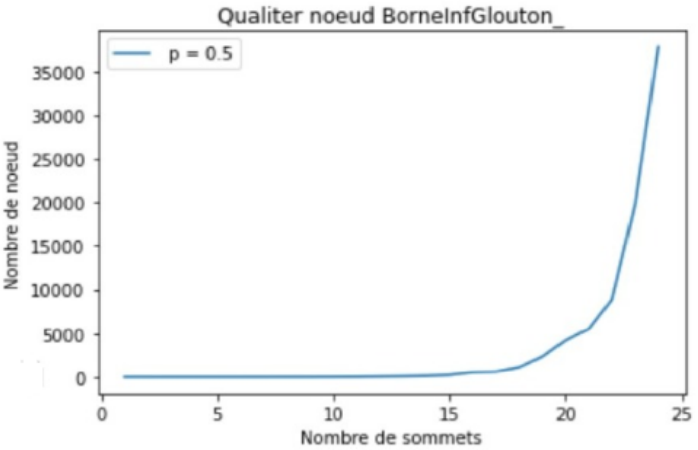






Comme vu dans la partie III, l'algo couplage renvoyait une réponse plus rapide que le glouton et donc en utilisant ces algos en tant que solution réalisable, on obtient un résultat **plus rapidement avec le couplage.** En effet, d'après les 2 premiers graphiques du haut, on voit que pour n = 23 alors t **= 4 sec** pour le couplage et t **= 8 sec** pour le glouton, soit le double. De plus, étant donné qu'on obtient une courbe exponentielle pour les graphes log / log et une droite affine pour les courbes en log / n , on a donc une **complexité exponentielle.**





Ici, on compare le nombre de nœuds générées et on remarque que l'on génère environ **le même nombre de nœuds** dans les 2 cas , pour **n = 23** on a ~ 30 000 nœuds dans les 2 cas.

⇒ **Comparaison entre ces algos et branchement simple :**

Pour n = 15 et p = 0.5: on a pour le branchement simple un temps d'exécution d'environ *2 sec et 10 000 nœuds* générés alors qu'avec la méthode des bornes on a un temps voisin de *0 sec et 1000 nœuds* générés.

Pour n = 19 et p = 0.5: on a pour le branchement simple un temps d'exécution de *11 sec et 200 000 nœuds* générés alors qu'avec la méthode des bornes on a un temps voisin a *0 sec et environ 2500 nœuds* générés.

Cela montre bien que l'ajout du calcul (en chaque nœud) d'une solution réalisable obtenue avec l'algorithme de couplage et le calcul d'une borne inférieure basée sur la relation en IV.2 **diminue considérablement le temps d'exécution et le nombre de nœuds générés.**

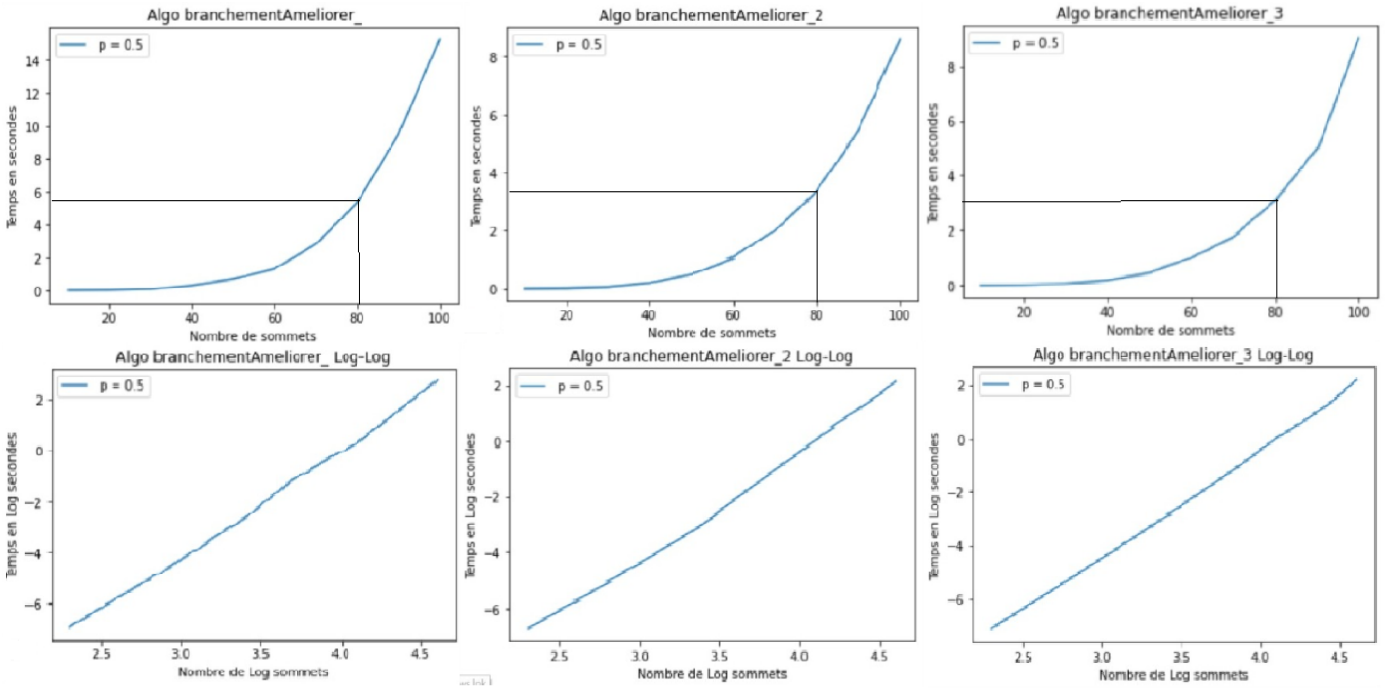
Couverture de Graphe, VERTEX COVER.
ABITBOL Yossef, 3804139,
SERRAF Dan, 3971120.

3) Améliorations de Branchement:(3 étapes) :

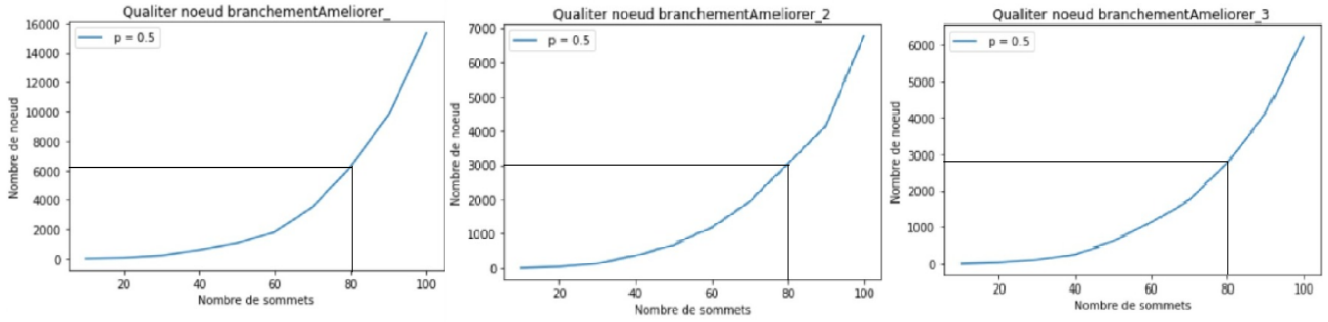
1. Dans la 2eme branche, on **prend v** dans la couverture mais **pas u** car le cas où on le prend étant traité dans la première branche, on doit alors prendre tous les voisins de u (et on les supprime du graphe). AMELIORER_

2. Afin d'éliminer un maximum de sommets dans la deuxième branche, il semble intéressant de choisir le branchement de manière à ce que le sommet **u soit de degré maximum** dans le graphe restant. AMELIORER_2

3. Dernière amélioration, on supprime tous les sommets de degré 1 sans les brancher. AMELIORER_3



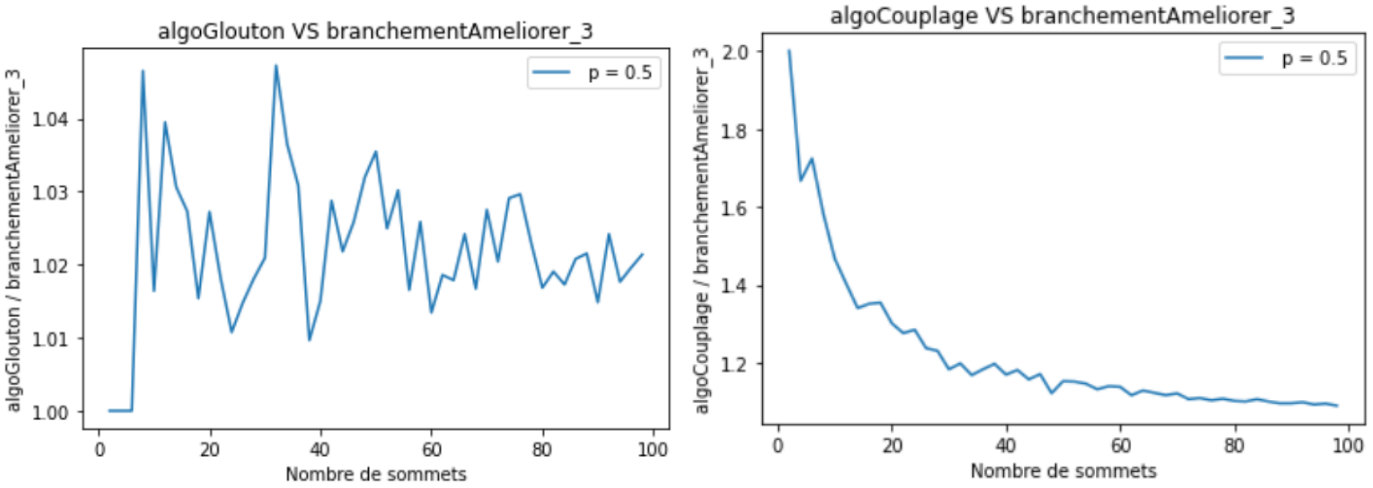
On compare les 3 améliorations décrites plus haut, pour n = 80 : on a pour AMELIORER_ un temps d'environ **5 sec**, pour AMELIORER_2 **3.3 sec** et AMELIORER_3 **2.9 sec,** cela montre que **Ameliorer_3 renvoi le résultat le plus rapidement possible** suivi de très près par ameliorer_2 et enfin ameliorer_. Donc, le fait de supprimer tous les sommets de degré 1 sans les brancher améliore de peu le temps d'exécution. De plus, les 3 graphiques en log / log affichent les 3 une droite, synonyme de complexité polynomiale de degré : [4.184 , 3.85 , 4.054] **soit 4.**



Enfin, on compare le nombre de nœuds générés, pour n = 80 : on a respectivement, **6100, 3000 et 2800 nœuds** générés. La courbe a une allure exponentielle pour les 3 figures.

4) Qualité des algorithmes approchés :

Une fois tous les algos codé, on veut maintenant comparer les algos approchées et ceux de branchement afin d'obtenir **un rapport d'approximation**.



Le rapport d'approximation de algo Glouton avec le branchement le plus amélioré est d'**environ 1.02** avec un **pire rapport de 1.05** obtenu pour n= 10 et 30.
Le rapport d'approximation de algo Couplage avec le branchement le plus amélioré **diminue progressivement en fonction de n**, pour un n faible (0-10) on a un pire rapport de **2**-1.7, plus **n augmente plus le rapport converge a 1.**

CONCLUSION:

- Le premier algo, branchement simple génère **énormément de nœuds** (800 000 pour n=20) et un temps d'exécution de 6 sec, d'une complexité exponentielle.
- Pour diminuer le nombre de nœuds généré on a donc implémenté la méthode des bornes en utilisant les algos de Couplage et Glouton. toujours de complexité exponentielle, on a comparé avec le premier branchement et on a montré que cette méthode permettait de **diminuer de beaucoup le nombres de noeuds** et **le temps d'exécution MAIS** à partir d'un certain n (pour nous 25) cela devient très compliqué de pouvoir tester l'algo du par son temps et son nombre de noeuds.
- On a donc procédé aux améliorations de branchement **en 3 étapes.** Grâce à ça, on a pu **augmenter considérablement notre limite de n, notre temps d'exécution, le nombre de nœuds générés** et on est passé d'une complexité exponentielle a une **complexité polynomiale.** Par exemple, pour n=100, on obtient de meilleur résultat que pour n=25 avec juste l'ajout des bornes.
- Enfin, le rapport d'approximation entre le meilleur branchement et Glouton est très faible **~1.02** et pour Couplage, il commence avec un rapport de 2 avec un n= 0 mais **converge a 1** n->∞