SCIENCES SORBONNE UNIVERSITÉ

MASTER DATA SCIENCE- DAC

RAPPORT DE PROJET COMPLEX

Couverture de graphe

MU4IN900, Master Informatique

REALISER PAR:

DJEDDAL Hanane.

TOUZARI Liticia.

2019/2020

Sommaire

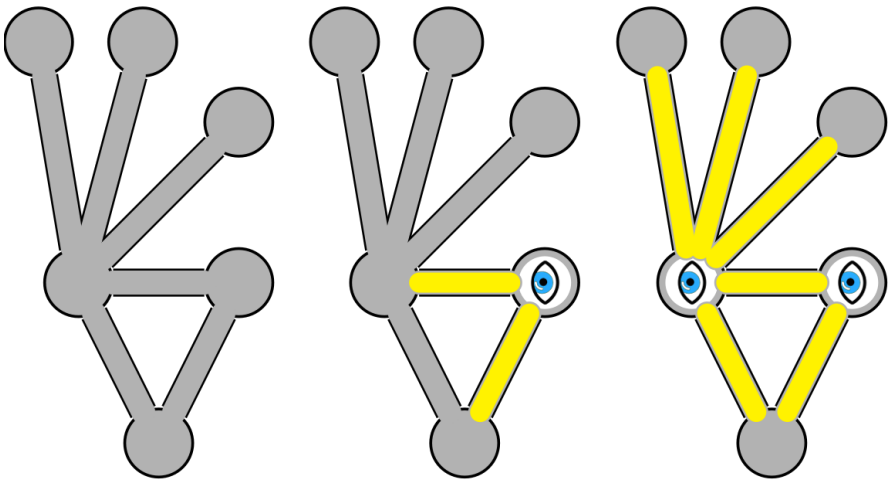
Table des matières

INTRODUCTION

Le problème de couverture minimum par sommets (Vertex Cover) est un problème classique en théorie des graphes. Il s'agit, étant donné un graphe G= (V, E) non orienté où V est l'ensemble des n sommets et E l'ensemble des m arêtes de G, de trouver un ensemble minimum V' ⊆ V tel que toute arête e de E a au moins une de ses extrémités dans V.

Le problème de décision associé à ce problème est NP-complet, et fait partie des 21-problème NP-complets de Karp.

Dans ce projet, on va implémenter différents algorithmes pour résoudre ce problème, et les comparer selon des tests sur différentes instances.



1 Graphe- Représentation et manipulation :

1.1 Représentation :

Dans le cadre de ce projet on va représenter un graphe par un tableau de listes d'adjacences. Au niveau de l'implémentation, une instance du graphe est un dictionnaire python qui associe à chaque sommet 'a' (la clé) la liste de ses voisins (la valeur).

NB : cette implémentation entraine qu'une arête est représentée deux fois.

1.2 Opérations de base :

Dans cette section, on va définir l'ensemble des méthodes de base qui vont nous permettre d'implémenter les différents algorithmes :

creer_graphe (nom_fichier) : construit le dictionnaire représentant le graphe à partir d'un fichier ayant le format :

Nombre de sommets
n
Sommets
[liste des sommets séparés par un retour chariot]
Nombre d aretes
m
Aretes
[liste des aretes séparées par un retour chariot]
supprimer_sommet(G,s) :retourne un nouveau graphe obtenu à partir de G en supprimant s. G n'est pas modifié.
supprimer_ens_sommet(G,ens) :retourne un nouveau graphe obtenu à partir de G en supprimant un ensemble de sommets ens. G n'est pas modifié
get_degre(G): retourne un dictionnaire qui associe à chaque sommet son degré.
get_degre_max(G): retourne une liste des sommets de degré max.
get_nbre_aretes: retourne le nombre des arêtes dans le graphes
is_empty (g): retourne vrai si le graphe n'a pas d'arêtes, faux sinon.

1.3 Génération d'instances :

Afin de tester nos algorithmes, il est nécessaire de générer des instances de graphes aléatoires, pour cela on a une méthode prenant en entrée N (nombre de sommets) et une probabilité p ($0<p<1$) qu'une arête appartienne au graphe, entre autres p est le paramètre d'une loi binomial.

Concrètement, on génère un dictionnaire de N sommets, et pour chaque arête possible, une binomial de paramètre p détermine si cette arête appartient au graphe, dans ce cas l'insérer dans le dictionnaire, sinon passer à la suivante.

2 Méthodes approchées :

Trouver un couplage maximal dans un graphe, un couplage étant un ensemble d'arêtes n'ayant pas d'extrémités en commun, est une des méthodes simples permettant de trouver une couverture. Or cette dernière n'est pas toujours minimale.

Dans cette partie, on va implémenter deux algorithmes, un exhaustif et l'autre glouton, de recherche d'un couplage maximal. Ces algorithmes vont nous permettre dans la suite, de trouver une solution non-optimale, facile à calculer qui va servir comme une borne dans notre recherche de la solution optimale.

1.1 L'algorithme glouton :

a) – Montrons que l'algorithme n'est pas optimal :

Procédant par le raisonnement par contre-exemple :

Soit I une instance du problème telle que G(V,E), V={0,1,2,3,4} et E={0-1,1-2,2-3,3-4}

G : 0 ——— 1 ——— 2 ——— 3 ——— 4

Dans une première itération, les sommets de degré max (= 2) sont : 1,2,3. Si l'algorithme choisit le sommet 2 (car il n'y a aucune contrainte sur la sélection), on aura, dans la deuxième itération, les sommets de degré max (= 1) : 0,1,3,4. S'il prend, cette fois-ci, le sommet 0, le résultat final sera la couverture {2,0,4} de cardinalité 3, or la solution optimale est {1,3} de cardinalité 2.

b) - Montrons que l'algorithme n'est pas r-approché:

On généralise le contre-exemple précédent pour montrer que l'algorithme n'est pas r-approché :
Soit r un réel, il existe toujours une instance I du problème telle que Val(glouton(I)) ≥ r x opt(I).
En effet, on prend G une chaine de n sommets (construite comme l'exemple précédent).

La solution optimale consiste à ne pas prendre la première extrémité de la chaine et d'alterner entre les sommets par la suite, alors opt = [n/2].

On construit I de tel sort à ce qu'on prend, en premier, le sommer [n/2]+1 qui est de degré max (= 2), ce qui va générer 2 sous chaines, on applique le même prince sur ces deux sous-chaines en choisissant le sommet [n'/2]+1 à chaque itérations.

Si n est pair alors Val(glouton(I)) = opt =[n/2]

Si n est impair alors Val(glouton(I)) = opt+1 =[n/2]+1

Alors on a un rapport d'approximation : (n/2) +1 / (n/2)=1 +2/n
Si on prend n=2/r, on aura un rapport de 1+2*r, alors on peut conclure que l'algorithme n'est pas r-approché

1.2 Comparaison :

a) – Complexités Théoriques :

L'algorithme couplage fait exactement m (nombre d'arêtes) itérations, et à chaque itération, il fait une recherche en O(n). Alors il a une complexité O(m*n)

Le glouton fait, dans les pires des cas, m itérations, et à chaque itération, il calcule le degré max O(n) et il supprime le sommet. Alors il a une complexité O(n*m).

b) – Complexités Expérimentales :

Comme la complexité est en fonction de nombre des arêtes et des sommets, on teste les algorithmes en variant p (la probabilité de présence des arêtes) et n.

En fonction de n :

i)-Couplage :

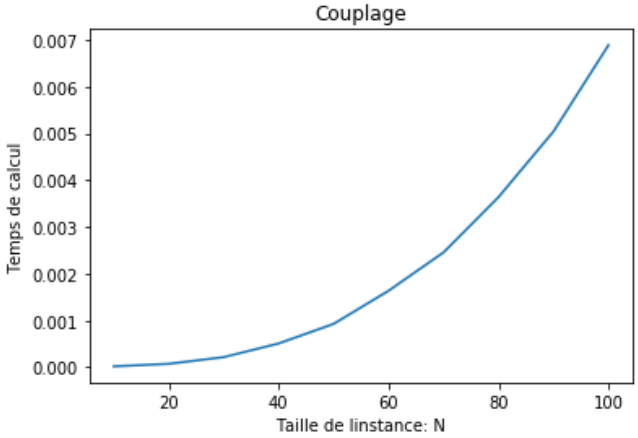


Figure3.2

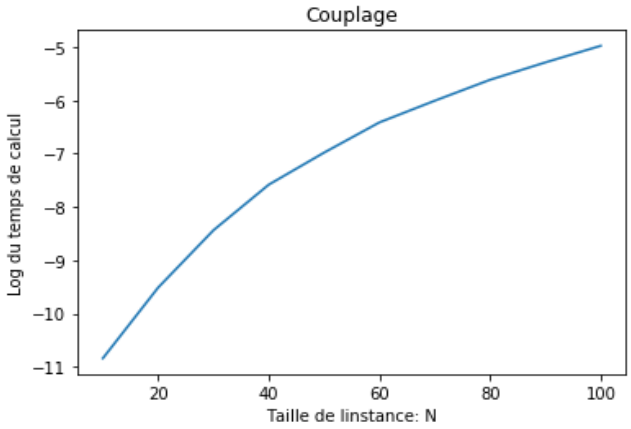


Figure 3.1

Figure 3.3

A partir du graphe 3.2 on constate que log(temps)= i*log(N), alors on peut déduire que temps= N^i. Pour avoir une approximation de i, on trace la courbe log(temps)=i log(N) en fonction de log(N), on obtient la droite de pente i (tracé 3). En moyenne i=2.7

ii)-Glouton :

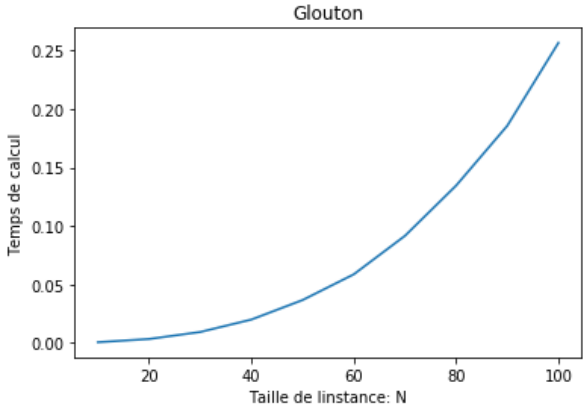


Figure 3.4

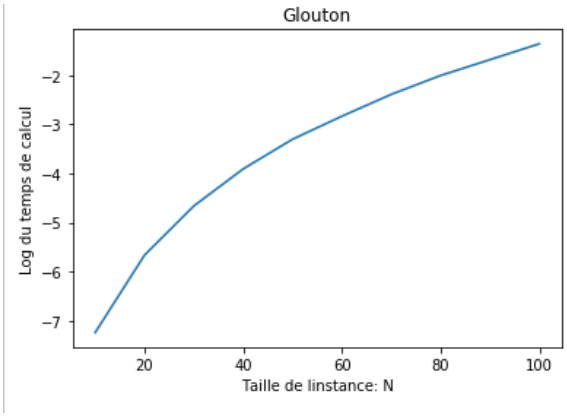


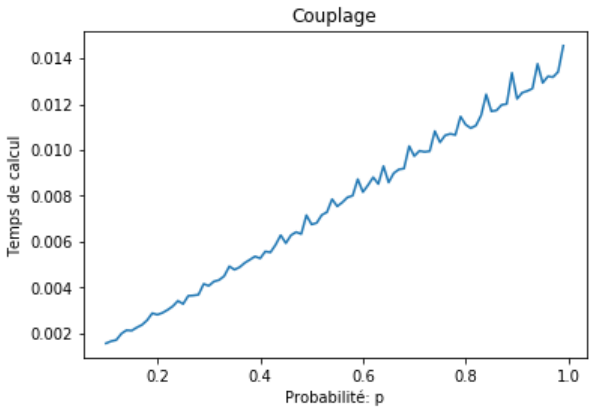
Figure 3.5

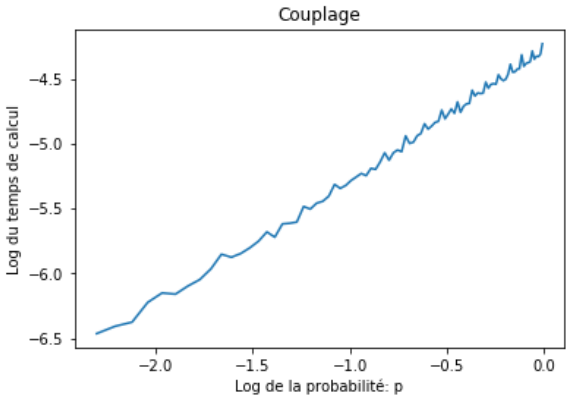
Figure 3.6

A partir du graphe 3.5 on constate que log(t)= 2*log(n), alors on peut déduire que t= n^2, ceux qu'on peut vérifier avec le graphe 3.6 qui nous donne une pente de 2.7

En fonction de p :

i)-Couplage :

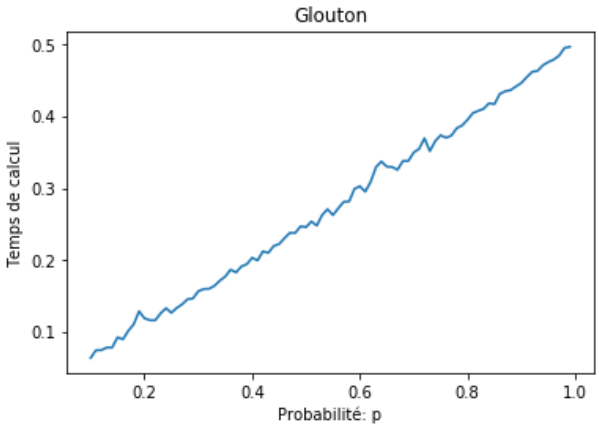


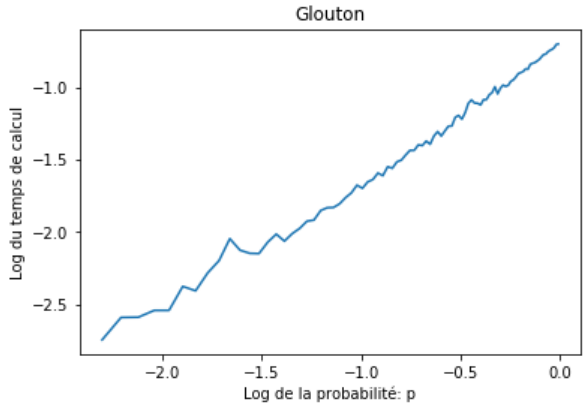



```
pente :  1.0498468549811444
```

A partir de ces figures, on voit bien que le temps évolue d'une façon linéaire en fonction de nombre d'arêtes, avec une pente = 1. Ce qui correspond bien à la complexité théorique.

ii)-Glouton :





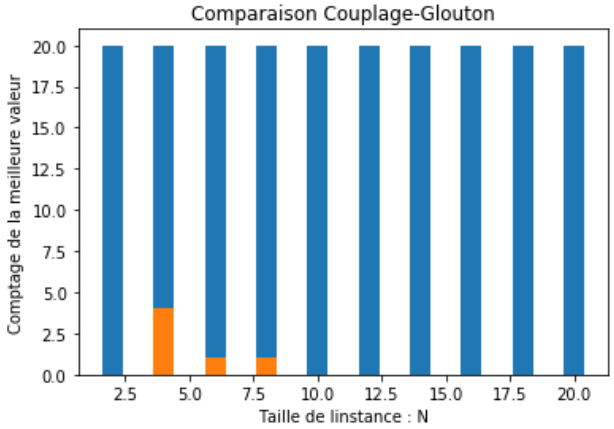

```
pente :  0.9486898752792463
```

De même, on voit bien que le temps évolue d'une façon linéaire en fonction de nombre d'arêtes, avec une pente légèrement plus petite que 1, car on ne parcourt pas toutes les arêtes. Ce qui correspond bien à la complexité théorique.

En générale, on peut remarquer que le couplage prend moins du temps (jusqu'à 15ms) que le glouton (jusqu'à 500ms), bien qu'en théorie, ils ont la même complexité. C'est dû au fait que le couplage fait des traitements de base (vérifier qu'un sommet n'est pas dans la liste : O(n)), alors que le glouton entraine des calculs supplémentaires à chaque itération : le calcul de degré max.

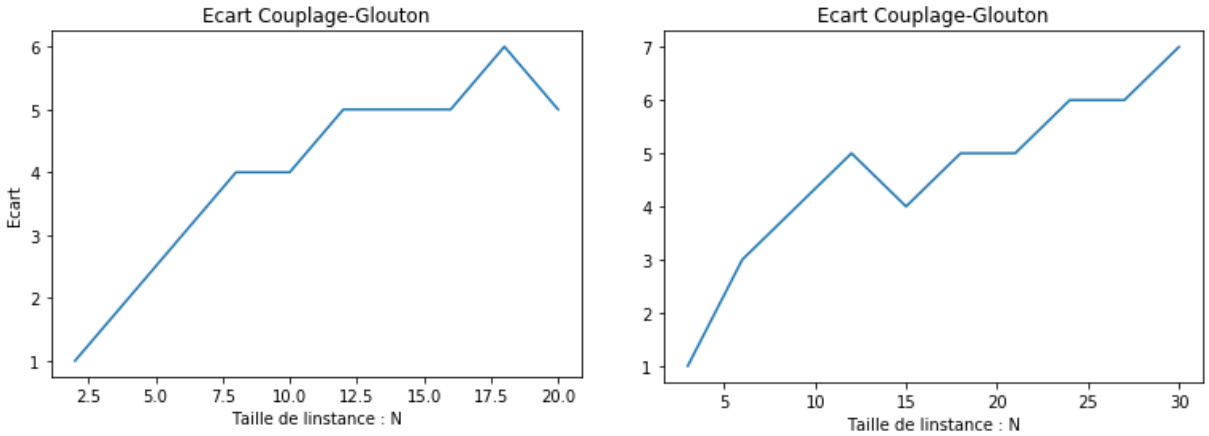
c) – Qualité de la solution :

Pour comparer les solutions retournées par les deux algorithmes, on fait une simple fonction qui, pour chaque valeur n, génère une vingtaine de graphes et calcule les solutions retournées par chacun des algorithmes. On compte, par la suite, les meilleures solutions retournées par le glouton et celles par le couplage. On trace le diagramme en bâtons suivant :



Le glouton retourne à chaque fois la meilleure solution entre les deux, et dans les cas où le couplage retourne une solution optimale, celle-ci est aussi retournée par glouton (Val(glouton) = Val(couplage)).

On pourrait aussi penser à tester si l'écart entre les solutions est stable, pour ça, on trace la moyenne des écarts en fonction de n :



On constate que l'écart n'est pas stable, et qu'il augmente en fonction de n. C'est à dire pour n très grand, la solution retournée par le glouton (car à partir du 1er test, on sait que val(glouton) ≥ val(couplage)) est beaucoup mieux que celle retournée par le couplage.

3 Séparation et évaluation :

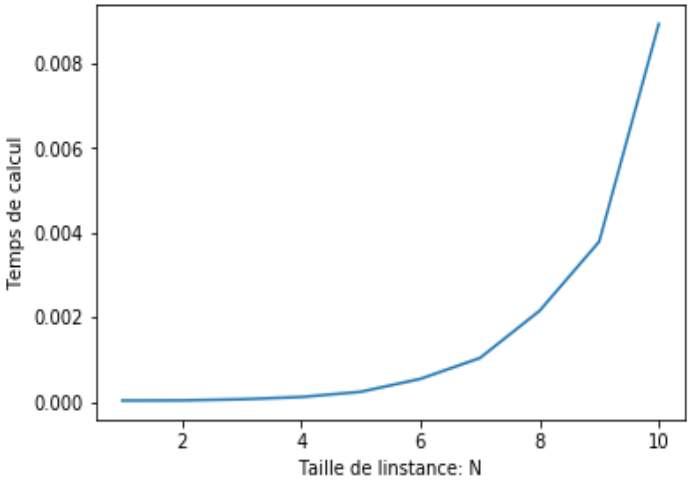
3.1 Branchement :

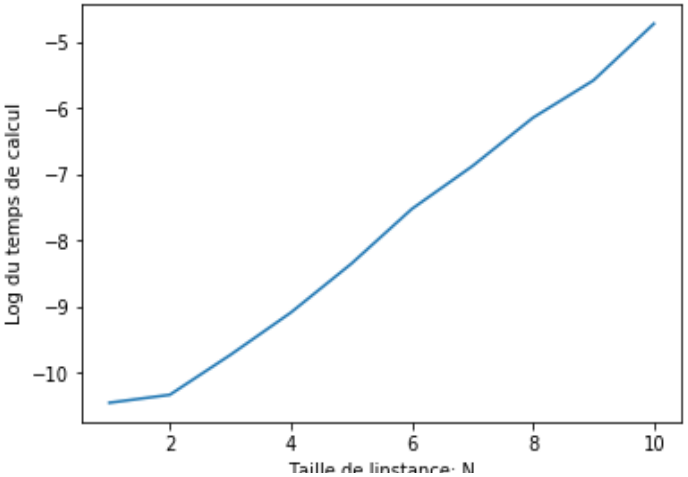
Partant d'un graphe G et une couverture C vide un nœud racine est créé, pour une arête e={u,v} de G :
-soit c'est u qui est dans la couverture C1 et donc un 1er nœud fils est créé auquel est associé un graphe G1 tell que G1 est obtenue à partir de G en supprimant l'arête e ainsi que les sommets incidents à u,
-soit c'est v qui est dans la couverture C2 et réciproquement un 2eme nœud fils est créé auquel est associé un graphe G2 tell que G2 est obtenue à partir de G en supprimant l'arête e ainsi que les sommets incidents à v,

Les 2 nœuds sont empilés dans une pile. A chaque itération un nœuds n est dépilé, une arête est tirée aléatoirement du sous graphe associé à n permettant ainsi la création de ses deux nœuds fils (empilés par la suite), tout en rajoutant chaque sommet à la couverture Ci qui lui est associé. On obtient ainsi toutes les couvertures minimales possibles du graphe G, on prend la meilleure.

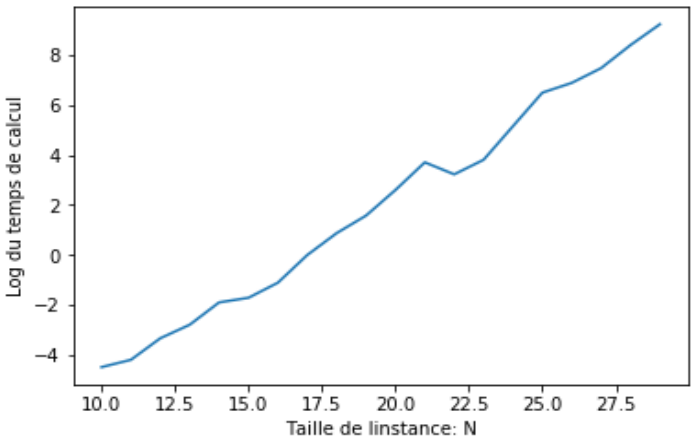
Complexité : L'algorithme construit un arbre complet, dans le pire des cas de profondeur n, il est donc de complexité 2^n

Tests : Pour Nmax=10 et p=0.5, on estime le temps de calcul moyen





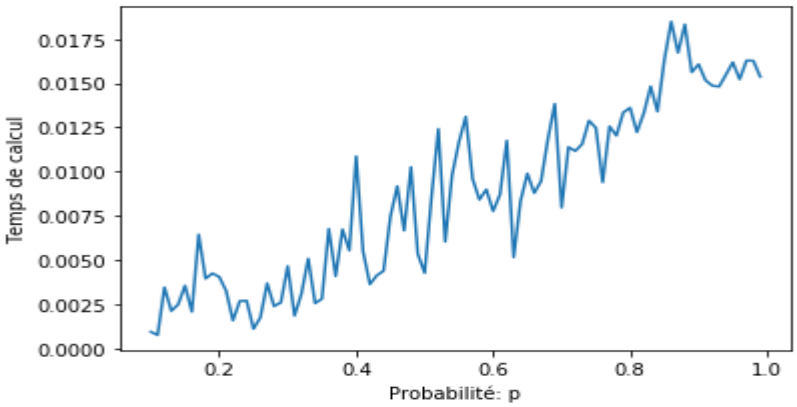
Pour Nmax=30



On voit bien log(temps de calcul) est une droite,

ce qui confirme bien que la complexité est bien

en 2^n (log(2^n)=nlog2 qui est une droite)



Le temps de calcul augmente linéairement avec l'augmentation du nombre arêtes, puisque la profondeur de l'arbre construit par l'algorithme augmente avec l'augmentation du nombre d'arêtes.

3.2 Ajout de bornes :

1) – Validité des bornes : prouvons que les bornes suivantes sont des bornes inferieures pour toute couverture C de cardinalité |C|:

   a. b1=$\left\lceil \frac{m}{\Delta} \right\rceil$ avec Δ le degré maximum des sommets du graphe

      Soit G un graphe de n sommets, m arêtes et Δ le degré maximum des sommets.

      Si « C » est la couverture minimale de G, alors chaque sommet de C couvre au maximum Δ arêtes dans le graphe G. Sachant que la sommes des arêtes que chaque sommet de C couvre est supérieur ou égale à m (sinon C ne serait pas couverture de G) concrètement $\sum_{i=0}^{|C|} mi \geq m$, (mi le nombre d'arêtes couvertes par le sommet xi dans C, donc mi est le degré de xi).

      Or on a que : |C|xΔ $\geq \sum_{i=0}^{|C|} mi \geq m$ donc |C|xΔ $\geq m$ et donc |C| $\geq \left\lceil \frac{m}{\Delta} \right\rceil$

   b. b2=|M| avec M le nombre de couplage

      Posons |M|=a, et supposons par absurde que |C|<a, C couverture de G

      Puisque a est le nombre de couplage dans M, alors M contient a arêtes de la forme e={xi,yi} telle que il n'y a aucune extrémité en commun avec ces arêtes autrement dit tous les sommets xi et yi sont différents (sinon ils n'auraient pas été sélectionnés par l'algorithme de couplage. Puisque le graphe G contient a arêtes complétements indépendantes il faudrait au moins a sommets pour les couvrir, donc toute couverture doit être de cardinalité supérieure ou égal à a, or |C|<a, ce qui est contradictoire.

      Conclusion : on a bien |C|$\geq |M|$

2) - Borne Avec Couplage :

On améliore l'algorithme de branchement en rajoutant des bornes pour limiter le nombre de nœuds explorés.

Pour la borne inférieure, on la calcule selon la question précédente, à chaque itération, le but étant de la faire rapprocher à la valeur réelle. Quant à la borne supérieure, on commence par la valeur retournée par l'algorithme de Couplage, et on la met à jour dans deux cas :

-En arrivant à une feuille qui donne une couverture meilleure que celle qu'on a.

-A un nœud donné, le couplage calculé sur le graphe restant retourne une meilleure solution que celle qu'on a.

3) - Borne Avec Glouton :

On procède de la même façon que 4.2, mais cette fois-ci, on calcule la borne supérieure en utilisant l'algorithme glouton.

Intérêt : On vient de voir que les solutions retournées par le Glouton sont d'une meilleure qualité que celles retournées par le Couplage. Ce qui permet de trouver des bornes supérieures plus proches à la solution optimale rapidement, et par conséquence n'explorer pas autant de nœuds.

4) – Sans borne supérieure :

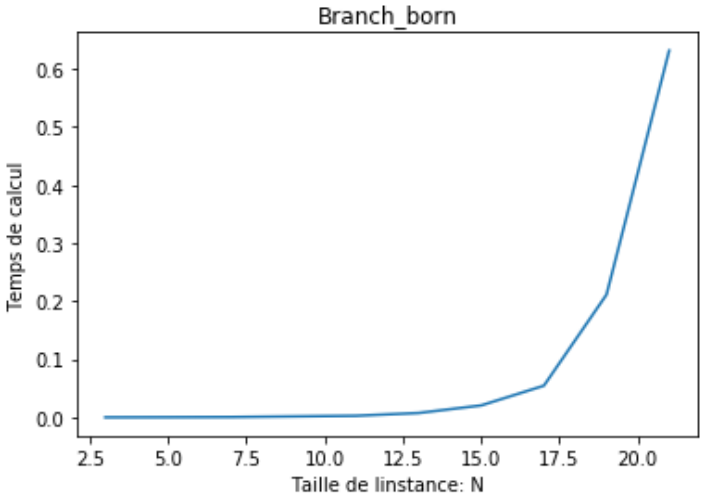
Dans ce cas, on se contente des bornes inférieures. Ce qui nous permettre de réduire le temps de calcul des bornes à O(1) au lieu de construire une solution complète à chaque itération. Mais on explore plus de nœuds.

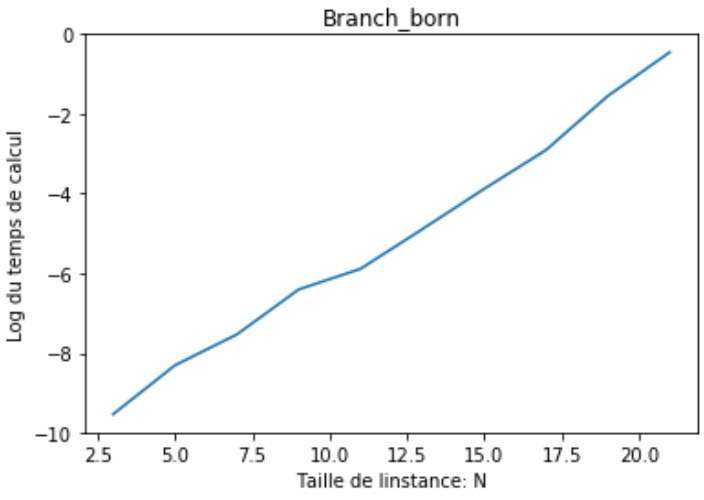
Complexité théorique :

La complexité en pire des cas reste toujours 2^n, mais on l'atteint rarement. On peut même la diminuer jusqu'à la moitié, ou mieux : arriver à la solution optimale sans aucune exploration. Ceci dit, les calculs supplémentaires des bornes restent non-négligeables dans les 2 premiers cas.

5) – Tests :

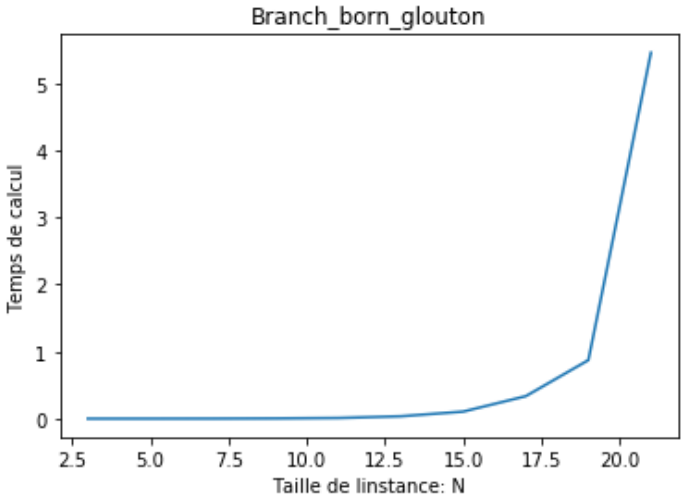
Borne Avec Couplage : Nmax=20, p=0.5

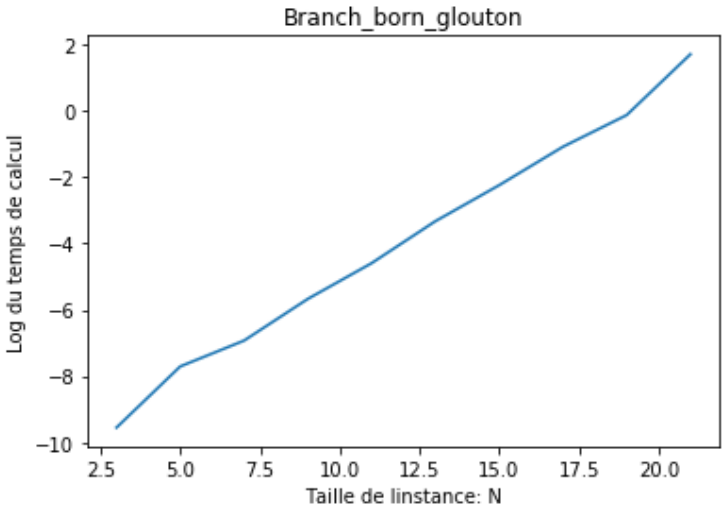




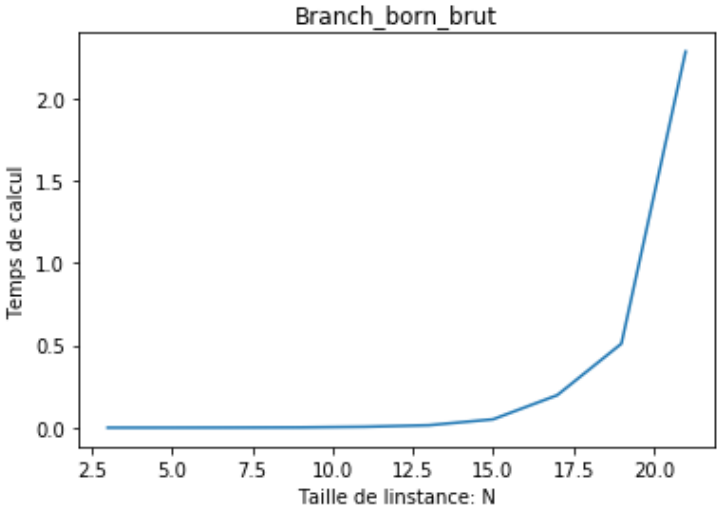
Complexité : On voit bien log(temps de calcul) est une droite, ce qui confirme bien que la complexité est bien en 2^n ($\log(2^n)=n\log 2$ qui est une droite).

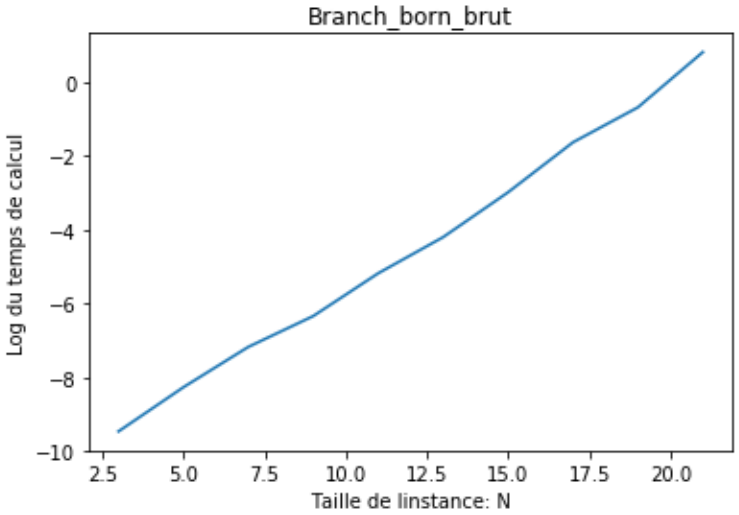
Borne Avec Glouton

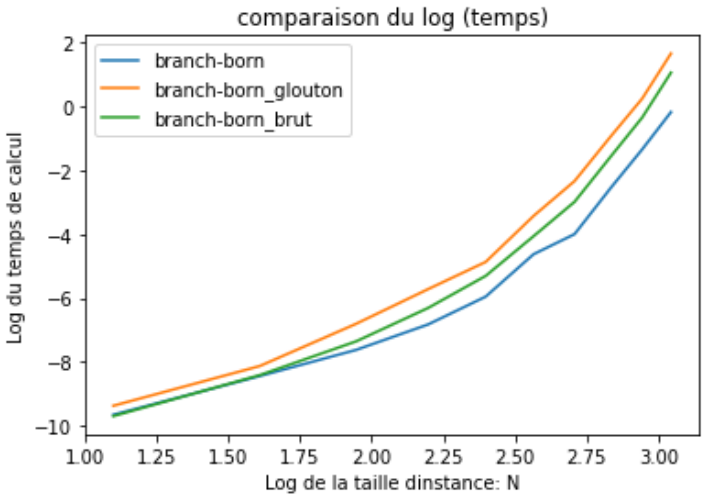


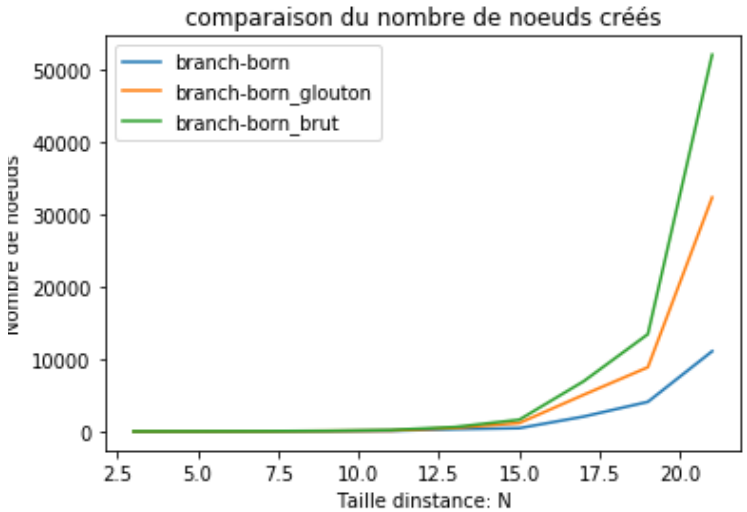


Sans borne supérieure









Comparaison :

Le branchement brut (sans borne supérieure) explore beaucoup plus de nœuds ce qui entraine un temps de calcul plus long, mais au même temps, l'absence des calculs des bornes supérieure améliore ce temps de calcul. D'une façon symétrique, le glouton explore moins de nœuds, mais le temps de calcul des bornes supérieure en O(m*n). Le Couplage fait un compromis entre les deux, et donne les meilleurs résultats (en terme du temps et nœuds explorés)

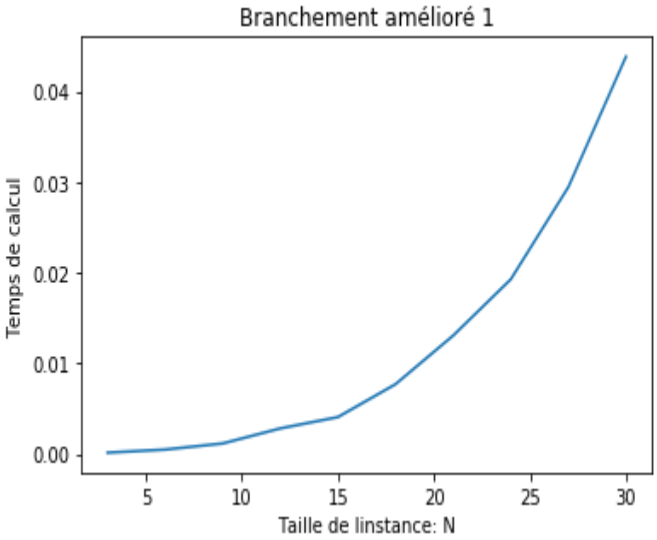
3.3 Amélioration du branchement :

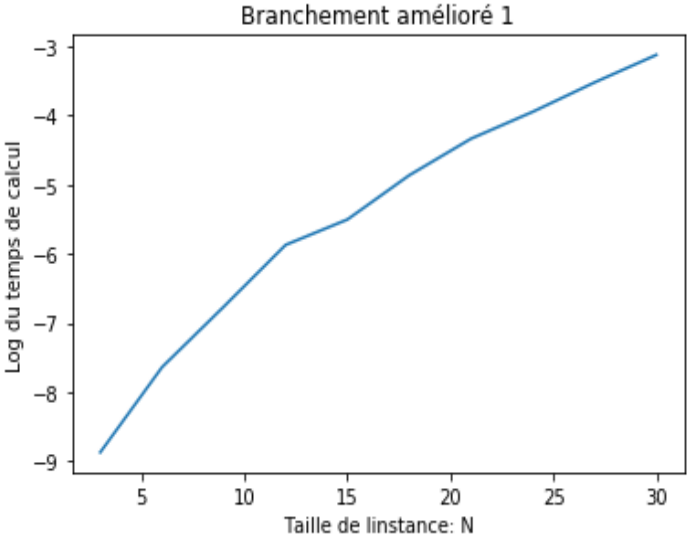
3.3.1

Lorsque une arête e={u,v} est branché, la branche ou l'on prend u est construite comme précédemment, par contre celle où c'est sommet v qui est rajouté à la couverture on ne traite pas le sommet u (on peut le supprimer du sous graphe associé à ce nœud) puisque ce cas sera traité dans la 1ere branche (c-à-d la possibilité d'avoir les deux sommets u et v dans une même couverture min) et dans ce cas tous les voisins de u sont rajouté à la couverture (puisque u n'étant pas dans la couverture il faut prendre ses voisins pour pouvoir couvrir les arêtes entre u et ses voisins).

Cet algorithme permet de renvoyer toutes les différentes couvertures minimales possibles (sans répétitions), il suffit de prendre à la fin la couvertures min optimale.

Tests : Nmax=30 , p=0.5





Complexité :

Le tracé de la courbe log(temps) en fonction de N (tracé 2) donne le tracé d'une fonction logarithmique

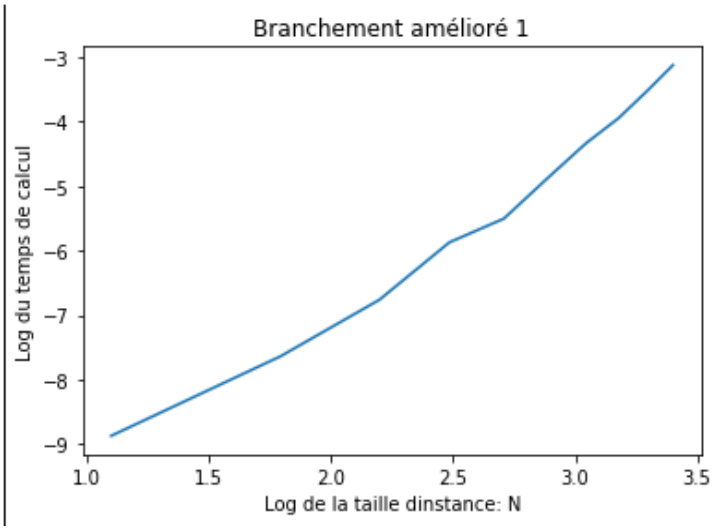
ie : log(temps)=i log(N)

Donc l'algorithme et de complexité $O(n^i)$

Pour avoir une approximation de i,

on trace la courbe log(temps)=i log(N) en fonction de log(N), on obtient alors une droite de pente i (tracé 3).

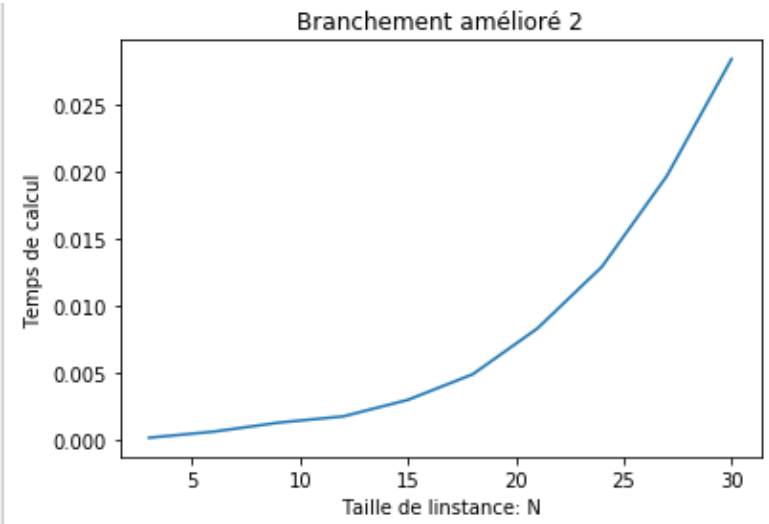
En moyenne i=3.7

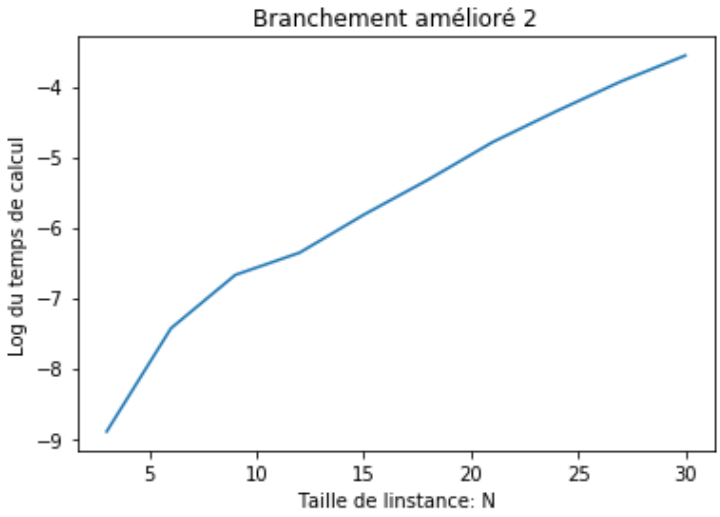


3.3.2

L'algorithme de branchement précèdent est modifié de sorte à ne plus tirer une arête e aléatoire mais plutôt une arête dont l'un des sommets (sommet u) est de degré maximal, cela permet de supprimer le maximum de sommet dans la deuxième branche (sous graphe associé au nœud qui considère v dans la couverture).

Test : Nmax=30 , p=0.5





Complexité :

Le tracé de la courbe log(temps) = i log(N) (tracé 2)

La complexité de l'algorithme est de $O(n^i)$

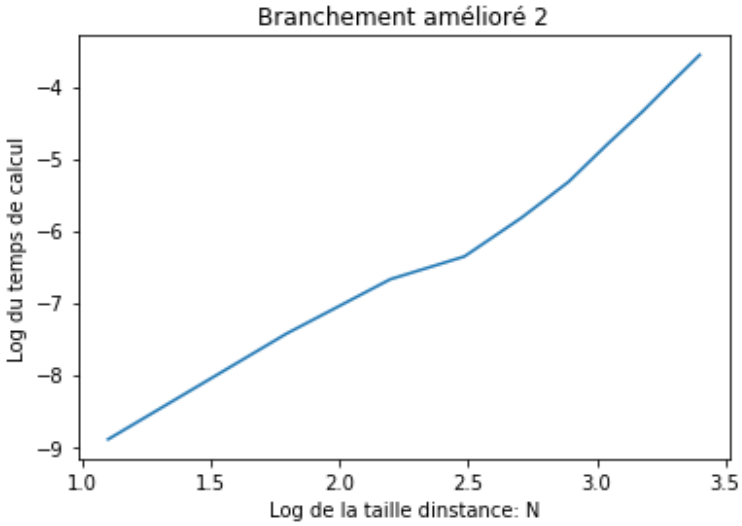
Pour avoir une approximation de i,

on trace la courbe log(temps)=i log(N) en

fonction de log(N), on obtient la

droite de pente i (tracé 3).

En moyenne i=3.2

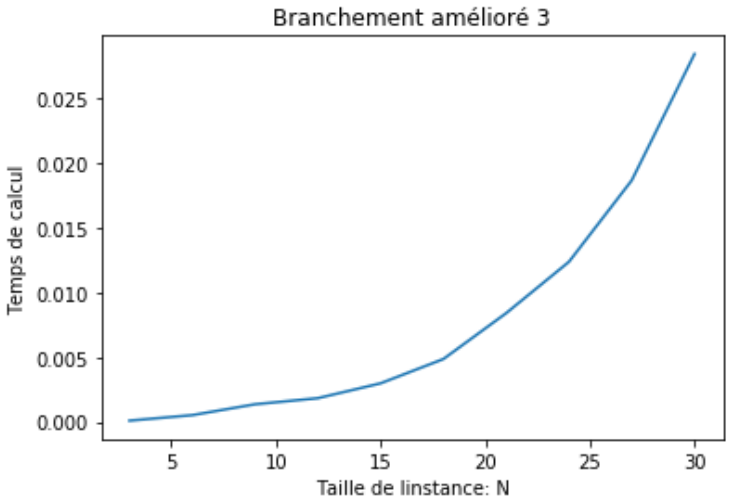


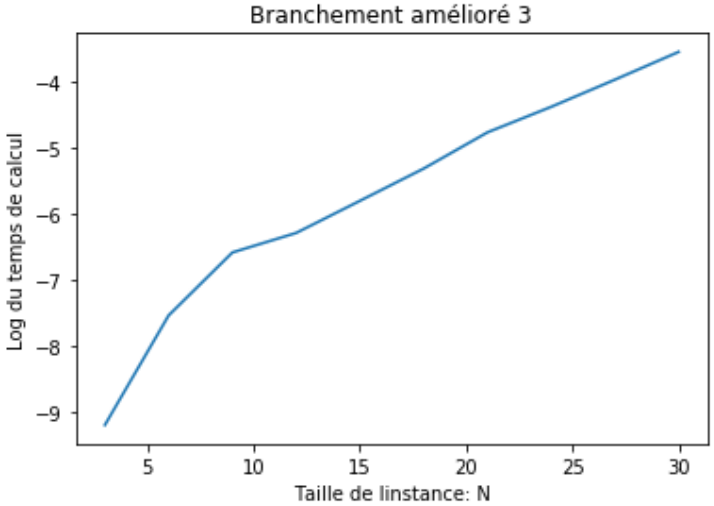
3.3.3

Dans un graphe G, si un sommet 1 est de degré 1, alors il existe toujours une couverture optimale qui ne contient pas u.

Supposons que C est une couverture optimale contenant u : un sommet de degré 1, soit v le sommet voisin de u, puisque la seule arête que permet u de couvrir est e={u,v} il suffit de remplacer le sommet u par v dans C', v couvre aussi l'arête e, C' est donc une couverture optimale.

Test : Nmax=30 , p=0.5





Complexité :

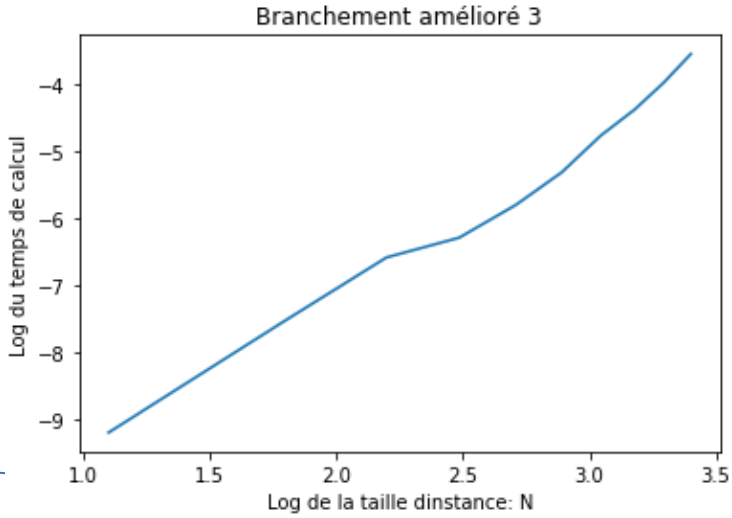
Le tracé de la courbe log(temps) = i log(N) (tracé 2)

La complexité de l'algorithme est de $O(n^i)$

Pour avoir une approximation de i, on trace la courbe

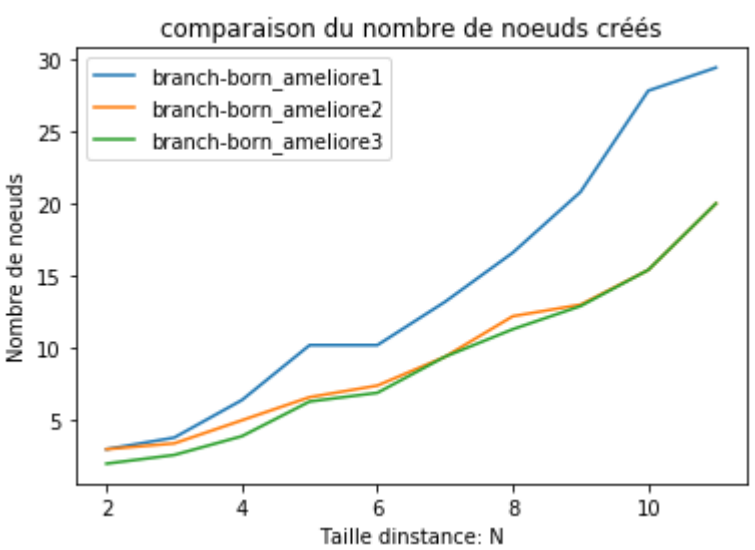
log(temps)=i log(N) en fonction delog(N),

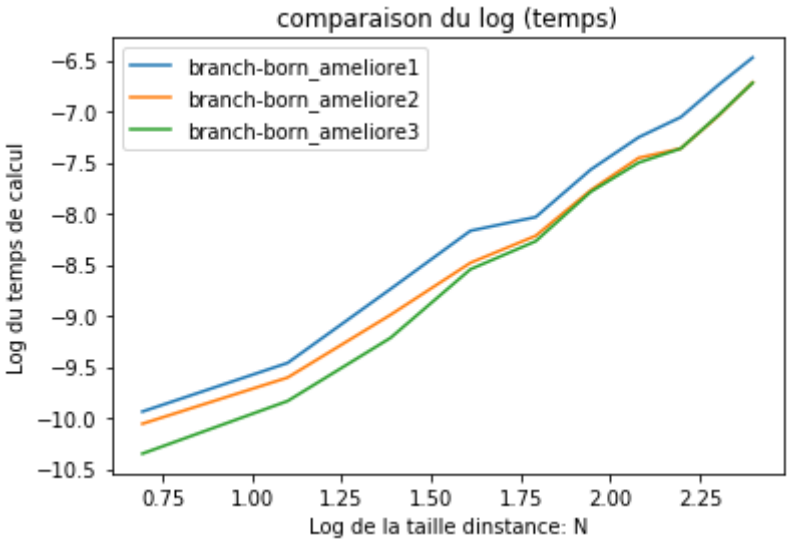
on obtient la droite de pente i (tracé 3). En moyenne i=3.1



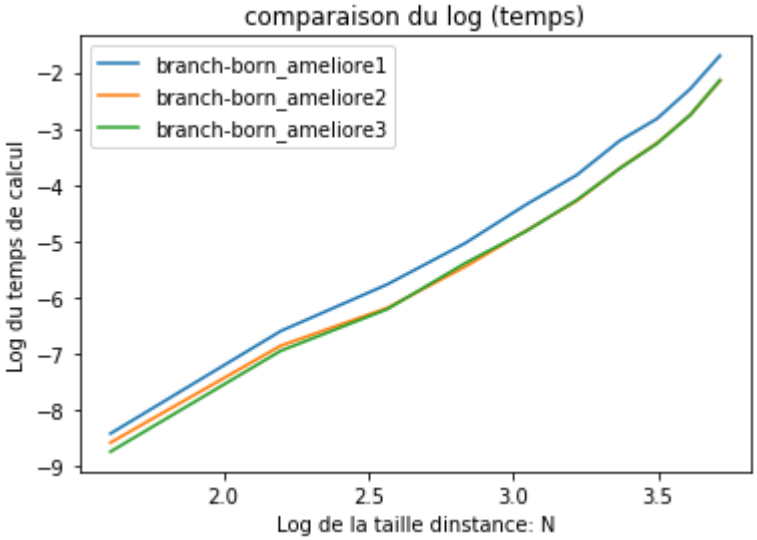
On constate que pour des tailles d'instances petites les algorithmes de branchement sont polynomiales, mais dés que la taille augmente la complexité de l'algorithme devient exponentiel $O(k^n)$, avec k<2.

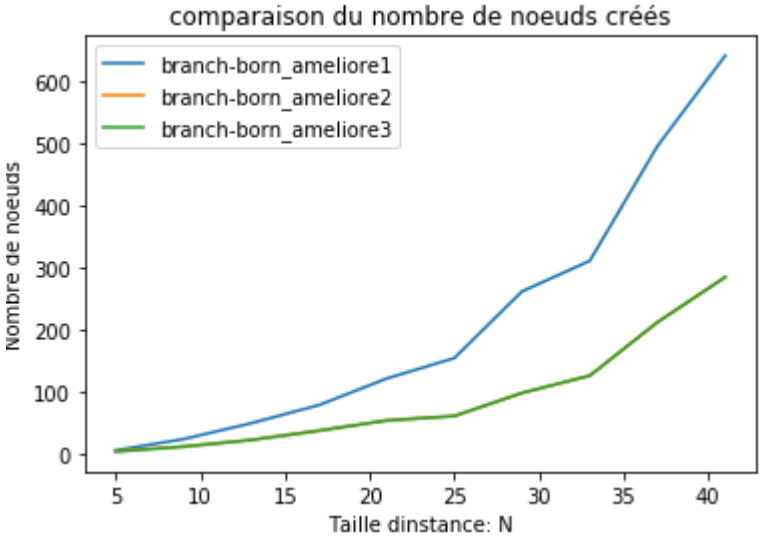
Comparaison : Nmax=10





Nmax=40 :

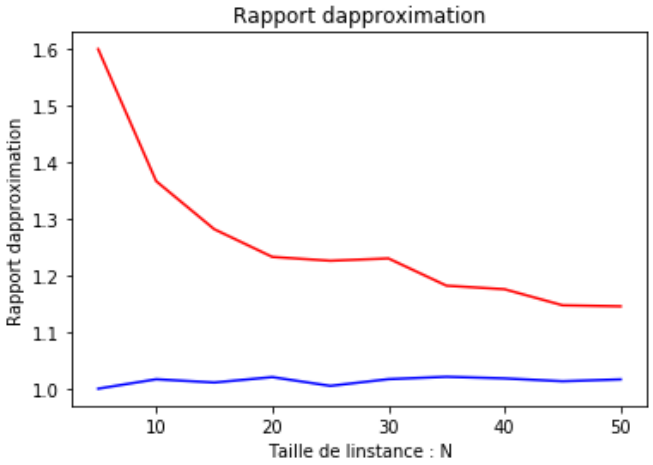




On constate bien que branch_born_ameliore3 est plus performant que branch_born_ameliore1 en temps de calcul et le nombre de nœuds générés, et légèrement plus performant que branch_born_ameliore1

3.4 Qualité des algorithmes approchés :

On calcule le rapport d'approximation entre sol(glouton)/sol_optimale et sol(couplage)/sol_optimale, on obtient :



Le rapport d'approximation pour le glouton est stable et a pour valeur un voisinage de 1. Par contre, le couplage varie en fonction de n, en rapprochant de 1 quand n augmente.

Conclusion : Dans le cas général, le glouton donne des solutions proches de l'optimum et ça pour toute valeur de n. Quant au couplage, on peut accepter ses solutions pour les grandes valeurs de n. Ce choix devient très critique pour des valeurs de n très grandes, car le temps de calcul de Glouton croit plus vite que le Couplage.

CONCLUSION

Les algorithmes étudiés dans ce projet pour la résolution du problème de couverture minimum par sommets (Vertex Cover) sont de complexité polynomiale, les plus simple et naïf sont de l'ordre de $O(2^n)$ ce qui reste long même pour de petites instances. Néanmoins on a pu améliorer les algorithmes de branchement pour avoir une exécution en temps polynomiale pour des instances de taille petite.

Le meilleur algorithme connu à ce jour s'exécute en $O(1.2738^k\ n)$. Ces algorithmes sont efficaces lorsque la taille de la couverture optimale est petite. Mais en pratique, ils ne sont plus utilisables dès lors que cette taille est grande, ce qui est malheureusement souvent le cas sur de gros graphes. Il est donc préférable d'utiliser des algorithmes approchés que des algorithmes exacts.