Projet COMPLEX - Rapport
Couverture de graphe

Bassem Yagoub & Ryan Ohouens

6 Novembre 2019

Table des matières

1 Introduction

Dans ce projet traitant du problème VERTEX COVER, nous allons nous intéresser à des algorithmes permettant d'obtenir des couvertures de graphes. Nous comparerons leur efficacité temporelle et les solutions retournées, puis nous verrons des méthodes d'optimisation permettant d'avoir ces résultats plus rapidement.

2 Graphes et outils utilisés

Afin de réaliser la structure de graphe et les fonctions demandées dans ce projet nous avons utilisé Python comme langage de programmation couplée avec la bibliothèque NetworkX qui donne accès à des graphes et leurs méthodes associées via des objets.

2.1 Opérations de base

Cette bibliothèque nous a permis de gagner un peu de temps dans le codage des opérations de base car elle est facile d'utilisation, mais nous verrons qu'elle est aussi efficace et qu'elle permet surtout d'avoir facilement un rendu graphique des graphes que l'on crée.

2.2 Génération d'instances

Notre fonction permettant de créer des instances de graphes en fonction d'un nombre de sommets n et une probabilité d'apparition d'arête p, prend également en paramètre un booléen déterminant si l'on veut afficher le graphe après création ou non. Ce booléen a également été ajouté en paramètre d'autres fonctions sur lesquelles cette option peut être utile.

En terme de temps d'exécution, nous avons les résultats suivants pour n=2500 :

- p = 0.00 : $\simeq$ 9s
- p = 0.25 : $\simeq$ 10s
- p = 0.50 : $\simeq$ 12s
- p = 0.75 : $\simeq$ 14s
- p = 0.99 : $\simeq$ 16s

On remarquera que, plus il y a d'arêtes, plus l'algorithme prend du temps, ce qui semble logique.

3 Méthodes approchées

Après implémentation des algorithmes glouton et couplage qui permettent de rendre une couverture d'un graphe, nous voulons comparer la qualité de leurs solutions retournées et leur temps d'exécution en fonction de n et p.

Nous allons tout d'abord voir que l'algorithme glouton n'est pas optimal avec le graphe visible ci-dessous, qui va servir de contre-exemple :

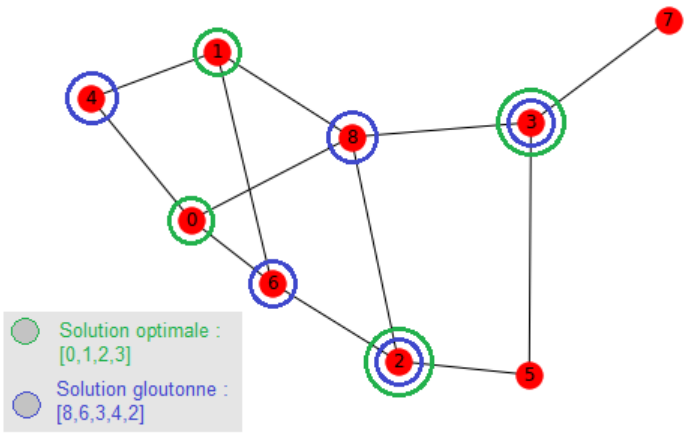

FIGURE 1 – Graphe pour lequel l'algorithme glouton ne rend pas une couverture optimale

On constate que la solution gloutonne retourne une solution de cardinalité=5 (en bleu) alors que la solution optimale a une cardinalité=4 (en vert). L'algorithme n'est donc pas optimal. Il n'est pas r-approché (pour un r aussi grand que possible) non plus, mais nous n'avons pas pu prouvé cela.

Pour comparer les deux algorithmes, nous avons créé une fonction qui va, pour chaque valeur de p (ici par exemple : {0.25, 0.50, 0.75, 0.99}), lancer les algorithmes sur des graphes de $n\_ites = 20$ tailles différentes. Les graphes auront des tailles qui augmenteront d'un coefficient $\frac{n\_max = 400}{n\_ites = 20} = 20$ sommets à chaque tour de boucle.

Afin d'avoir des résultats plus fiables, pour chacune de ces tailles de graphes et valeurs de p, elle exécutera les algorithmes sur $n\_moys = 5$ graphes créés aléatoirement et en fera une moyenne.
Comparons maintenant les résultats de l'algorithme glouton avec ceux de l'algorithme couplage.

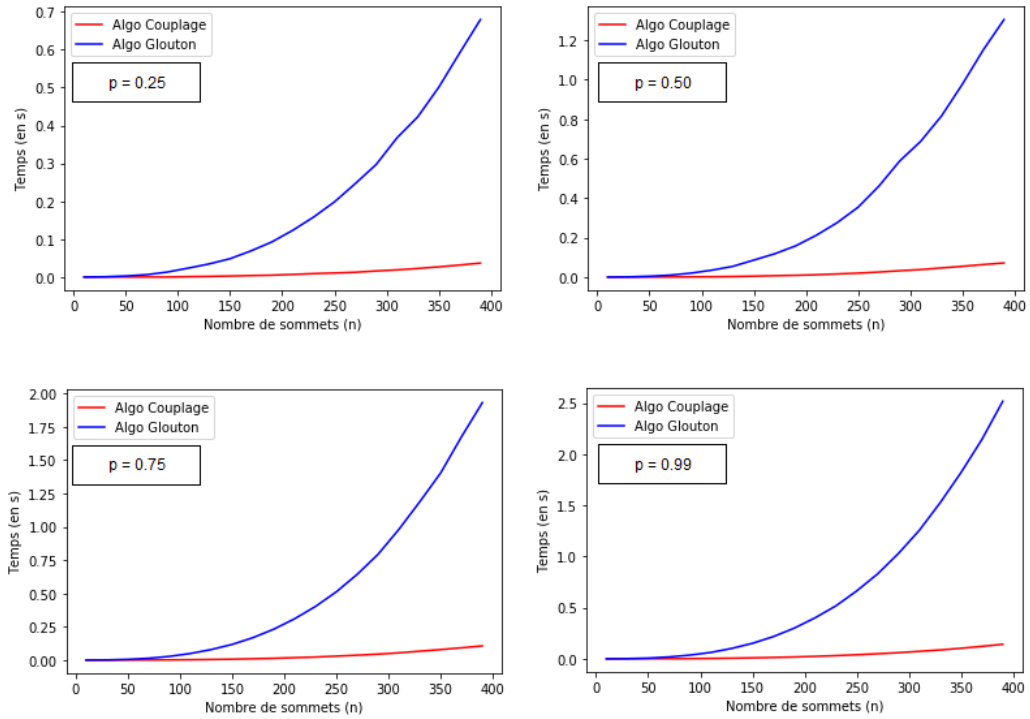

FIGURE 2 – Durées d'exécutions des deux algorithmes (n ∈ [10, 400], p={0.25, 0.50, 0.75, 0.99}))

Nous avons ci-dessus, les durées d'exécutions des algorithmes pour quatre valeurs de p différentes. On note à première vue que l'algorithme glouton est beaucoup plus lent que le couplage étant donné qu'il prend jusqu'à 2.5s contrairement à son "adversaire" qui ne dépasse jamais la demi seconde.
De plus, on remarque que, bien que les courbes aient des allures très similaires, les algorithmes et surtout le glouton prend de plus en plus de temps quand le nombre d'arêtes dans un graphe augmente.
Cela s'explique par le fait qu'il y a plus d'élements à visiter, donc trouver un sommet de degré max pour glouton, ou boucler sur toutes les arêtes pour couplage prendra plus de temps.

En ce qui concerne la qualité des solutions retournées par ces algorithmes, nous allons voir les résultats avec des graphes similaires, donnant les moyennes des cardinalités des couvertures retournées par les deux algorithmes, toujours en fonction de n=400 et p={0.25, 0.50, 0.75, 0.99}.

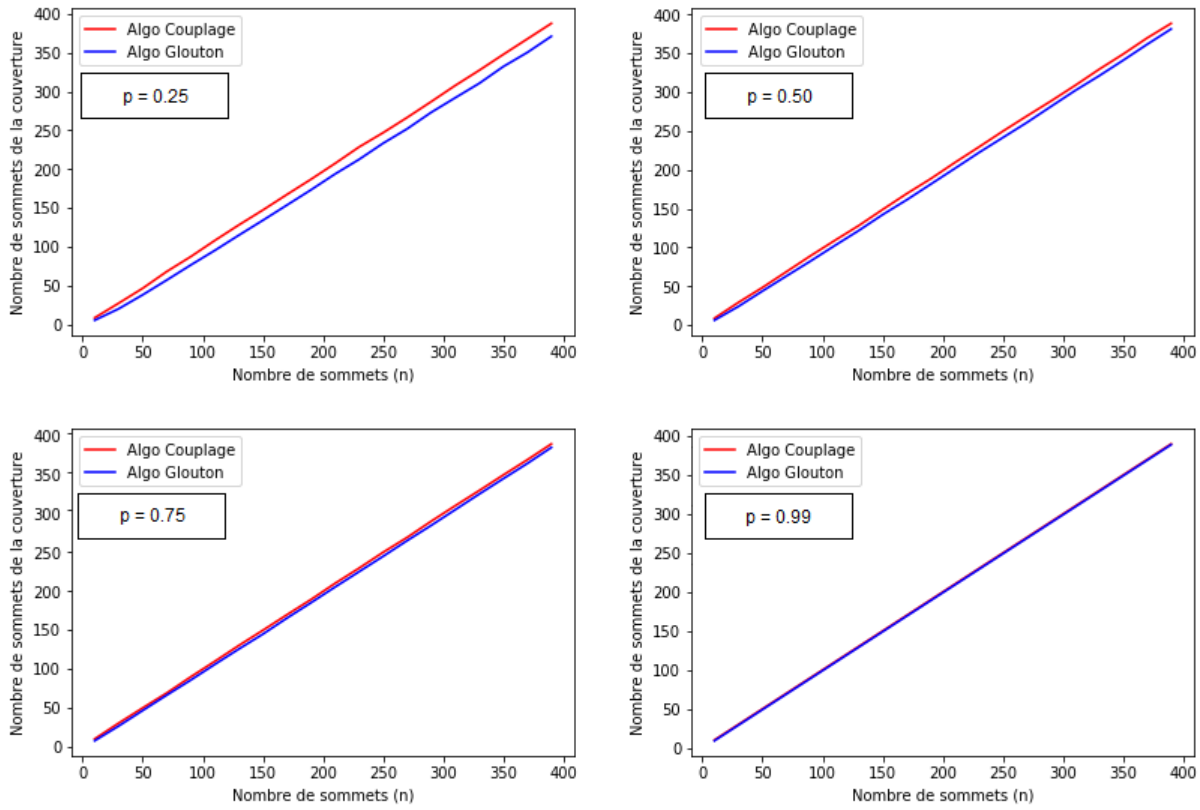

FIGURE 3 – Cardinalité des couvertures des algorithmes (n ∈ [10, 400], p={0.25, 0.50, 0.75, 0.99}))

L'impression que donnent ces résultats est que glouton est un peu plus efficace que couplage quand les graphes sont peu denses, mais plus ce sera le cas et moins il y aura une différence sur la solution.

4 Séparation et évaluation

Intéressons nous à présent à un autre algorithme qui est le Branch & Bound (B&B) qui retourne une couverture optimale d'un graphe. Le problème ne sera donc plus de savoir si la solution est la meilleure possible mais d'essayer de la trouver la plus rapidement possible.

4.1 Branchement

Premièrement, il faut implémenter l'algorithme. Pour cela, nous avons opté pour fonction récursive car cela nous paraissait plus naturel pour un parcours d'arbre. Elle va donc s'appeler elle même avec en paramètre les sous

graphes où on a retiré le sommet u et v d'une arête e, qui ira dans une couverture locale s'il y a encore au moins une arête dans le graphe. En remontant dans l'arbre, la fonction choisira la meilleure couverture.

Pour tester cet algorithme, nous l'avons lancé sur des instances de 2 à 20 nœuds avec un pas n=2,(on a donc 10 points sur nos graphes ci-dessous) et les deux valeurs extrêmes de p utilisées tout à l'heure : 0.25 et 0.99 car les courbes sont très similaires. Pour ces valeurs, nous avons évalué le temps et le nombre de nœuds visités par l'algorithme :

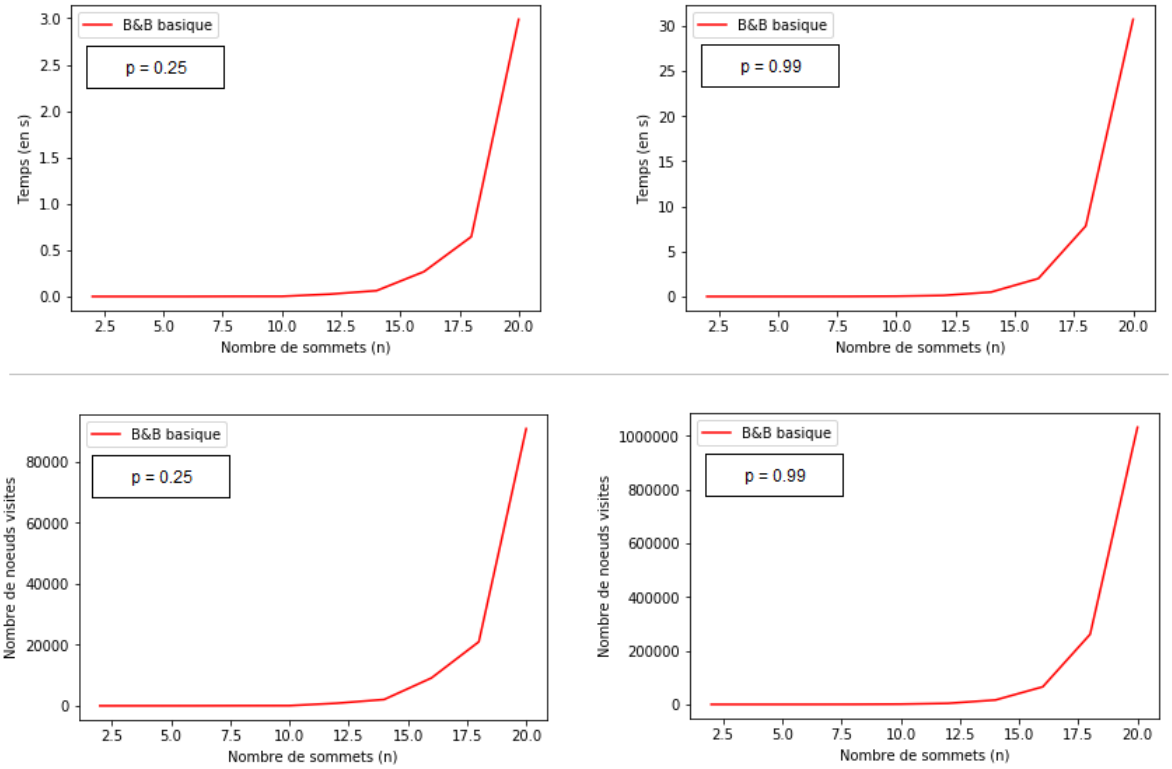

FIGURE 4 – Temps et nombre de nœuds du B&B (n ∈ [2, 20], p={0.25, 0.99}))

Comme on peut le voir, l'algorithme B&B est très lent, surtout par rapport à glouton et couplage, et il l'est de plus en plus quand n est grand. Sa durée est également d'autant plus longue quand p croît. Cela va de même pour le nombre de nœuds visités, on passe d'environ 90K nœuds à 1M.
On remarque que le nombre de nœuds tend vers 2^n quand p est proche de 1, ce qui est normal vu qu'on crée autant de branches qu'il n'y a d'arête dans chaque sous-graphe dans un arbre binaire de taille n (cela a été testé pour de nombreux graphes de petites tailles pour s'en persuader).

4.2 Ajout de bornes

L'objectif va maintenant être d'améliorer ces deux points noirs de l'algorithme avec l'ajout de bornes en premier temps. Commençons par montrer leur validité.
Montrons b1 :

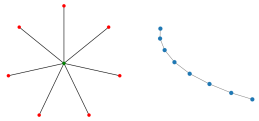

FIGURE 5 – Graphe étoile et Graphe ligne

Supposons l'existence de la couverture minimale de C et supposons qu'on a le graphe étoile, car c'est celui minimisant $\lceil \frac{m}{\Delta} \rceil$.
Supposons par l'absurde que $\lceil \frac{m}{\Delta} \rceil > |C_{min}|$, cela équivaut à $1 > |C_{min}| \Rightarrow |C_{min}| \geq 0$, donc qu'il n'existe pas de couverture.
Cela est contradictoire car il en existe une couverture minimale pour $|C| = 1$.
Par récurrence, si on modifie une arête du graphe étoile pour ne pas la placer sur le sommet(Δ), couverture+1 > borne+1, car $\Delta \geq 1$ et $\Delta \leq m$.
Donc, $|C_{min}| + 1 \geq (m+1)/\Delta$.

Supposons un graphe en ligne de n sommets tel que Δ=2, car c'est celui maximisant $\lceil \frac{m}{\Delta} \rceil$.
Supposons par l'absurde que $\lceil \frac{m}{\Delta} \rceil > |C_{min}|$.
Cela équivaut à $\frac{n}{2} > \frac{n}{2}$, ce qui est contradictoire.

Par récurrence, si on modifie une arête du graphe en ligne pour la placer sur le sommet(Δ), couverture+1 > borne+1, car $\Delta \geq 1$ et $\Delta \leq m$.
Donc, $|C_{min}| + 1 \geq m/(\Delta+1)$.
Il n'existe pas d'instance de graphe offrant de meilleur borne minimale ou maximale à $\lceil \frac{m}{\Delta} \rceil$. Donc pour toute instance de graphe, $|C_{min}| \geq \lceil \frac{m}{\Delta} \rceil$.

Montrons b2 :
Soit |M| le nombre d'arêtes du couplage M.
Si e ∈ M, alors e ∈ G, donc C est une couverture de G et de M. On obtient :
$\forall e \in M, \exists v \in C$, e est adjacente à V.
Ainsi :
$\forall e_1, e_2 \in M$ avec $e_1 \neq e_2$, $\exists\ v_1, v_2 \in C$ tel que e_1 est adjacente à v_1 et e_2 est adjacente à v_2.
De plus :
$e_1 \neq e_2$, elles ne partagent donc aucun sommet en commun car M est un couplage de G.
Donc : $v_1 \neq v_2$
On en déduit donc que, $\forall e \in M, \nexists v \in C$ tel que e est adjacent à v. Donc :
Si $|M| \geq 2$, alors $|C| \geq |M|$.
Si $|M| = 0$, alors $|C| \geq |M|$ car $|C| \geq 0$.
Si $|M| = 1$, alors pour $e \in M, \exists v \in C$ tel que e est adjacent à v. Donc $|C| \geq 1 = |M|$.

Montrons b3 :
On remarque que le maximum d'arêtes d'un graphe à n sommets est atteint quand le graphe est complet. Dans ce cas :
$m = \frac{n(n-1)}{2}$, donc $m \leq \frac{n(n-1)}{2}$
On pose :
$f : \mathbb{R} \to \mathbb{R}$
$x \to x^2 + (1\text{-}2n)x + 2m$

$f(|C|) = |C|^2 + (1\text{-}2n)|C| + 2m$
De plus : $|C| \leq n$, $m \leq \frac{n(n-1)}{2}$

Donc : $f(|C|) \leq n^2 + (1\text{-}2n)n + (n\text{-}1)n$
≤ 0
Donc : $f(x) \leq 0 \Leftrightarrow x \in [x1, x2]$ avec :

$$x1 = \frac{-((1-2n) - \sqrt{(1-2n)^2 - 8m})}{2} = \frac{-(2n-1-\sqrt{(2n-1)^2 - 8m})}{2}$$
$$x2 = \frac{-((1-2n) + \sqrt{(1-2n)^2 - 8m})}{2} = \frac{-(2n-1+\sqrt{(2n-1)^2 - 8m})}{2}$$

or $f(|C|) \leq 0$, donc $|C| \in [x1, x2]$ et donc $\frac{2n-1-\sqrt{(2n-1)^2-8m})}{2} \leq |C| \leq x2$

L'algorithme de branchement programmé précédemment retourne forcément une couverture de sommet mini-

male. En effet, supposons que cet algorithme retourne une couverture qui n'est pas minimale, cela voudrait dire que quelque part dans la construction de l'arbre, lors de la remonté on aurait choisi une branche avec une |couverture| < | couverture minimale|, or cela est impossible.

Lors de la construction de notre arbre les branches qui sont donc de longueur supérieurs à celles de la couverture minimale ne seront jamais prises. Sachant cela, on pourrait se décider de couper à ce moment précis pour économiser le nombre de nœuds parcourus, donc le temps d'exécution du programme.

Le but du jeu est de savoir, avant la construction de l'arbre, quelle sera la cardinalité d'une des couvertures minimales du graphe pour pouvoir définir les différentes coupes lors des branchements. Sachant que b_{Inf} = max{b1, b2, b3} :
On sait que $|C| \geq b_{Inf}$, de ce fait, lors des différents branchements, tant que $|C| < b_{Inf}$ on est certain qu'aucune couverture n'a été trouvée. On sait que $|C_{min}|$ est la plus petite valeur possible de couverture, donc on peut traduire notre relation précédente par $|C_{min}| \geq b_{Inf}$.

Dans notre arbre, quand on veut couper au nœud tel que $|C| \geq b_{Inf}$, 2 cas se présentent :
- $b_{Inf} = |C_{min}|$, donc notre coupe est juste
- $b_{Inf} < |C_{min}|$, on a donc coupé trop tôt, car on a pas eu la réelle cardinalité de la couverture minimale. Vu que la cardinalité de toutes les couvertures des branches coupées seront inférieures à $|C_{min}|$. Cela implique qu'on ne descendra jamais jusqu'aux feuilles de l'arbre ! On aura donc une couverture fausse.

Pour remédier à ce problème, l'ajout d'un $\varepsilon > 0$ à la borne inférieure permet de ne pas couper trop tôt. Ainsi on obtiendrait $|C_{min}| \leq b_{Inf} + \varepsilon$.
Dans notre arbre, quand on coupe, on souhaite obtenir ces 2 nouvelles situations :
- $b_{Inf} = |C_{min}|$, donc la coupe est juste
- $b_{Inf} > |C_{min}|$, on a coupé un peu tard mais on à économisé des nœuds car on est sur de tomber sur $|C_{min}|$ dans une autre branche.

Pour obtenir cette situation, il faut donc jouer sur le ε. S'il est trop petit, l'inégalité $|C_{min}| \leq b_{Inf} + \varepsilon$ ne sera plus respecté, cela reviendra à notre problème de départ. S'il est trop grand, on ne coupera jamais, cela reviendra au branchement sans coupe.

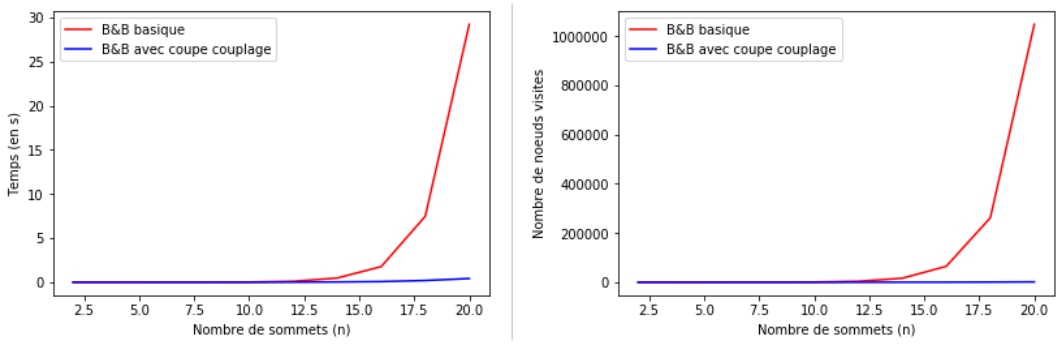

FIGURE 6 – Comparaison temps et nombre de nœuds visités entre B&B basique et avec coupe couplage

Comparons maintenant l'algorithme B&B avec sa version qui élague pour un p=0.99. On aperçoit un gain de temps et un nombre de nœuds visité dans l'arbre énorme, ce qui le rend beaucoup plus intéressant maintenant. Mais on peut encore l'améliorer :

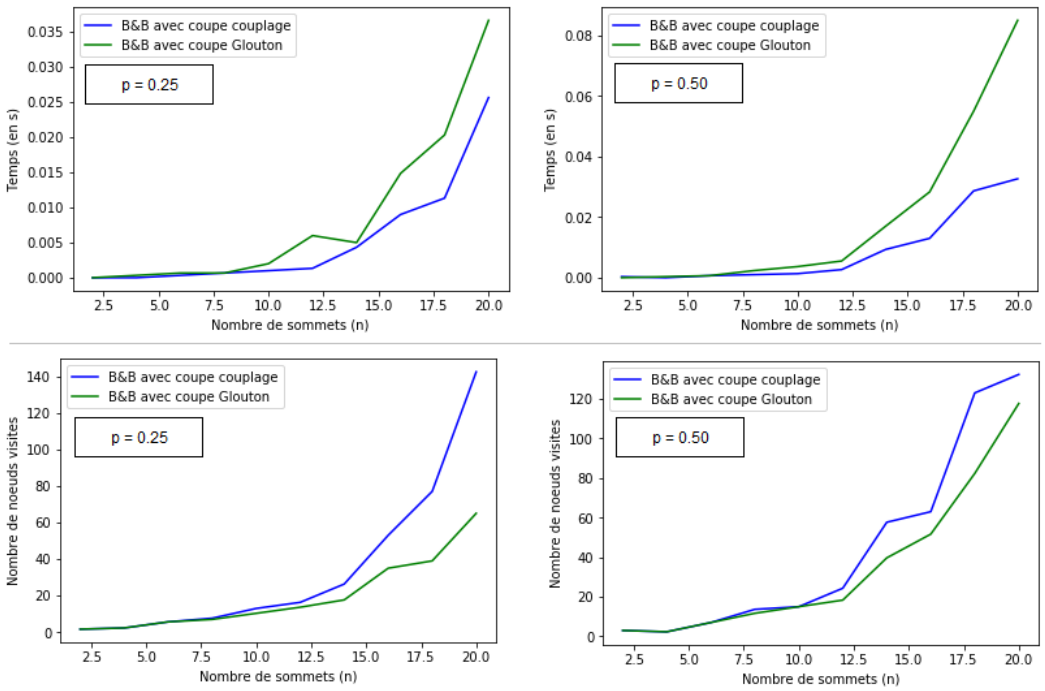

FIGURE 7 – Comparaison (temps et nombre de nœuds) entre coupe avec couplage et glouton

On peut constater que l'algorithme B&B version glouton prend plus de temps pour s'exécuter que l'algorithme couplage, on peut expliquer cela par le fait que notre implémentation de l'algorithme glouton était elle même beaucoup plus lente que couplage comme vu précédemment.
Cependant on peut remarquer que lors de la construction de l'arbre, l'utilisation de l'algorithme glouton permet de visiter moins de nœuds qu'avec l'algorithme couplage qui prend les sommets 2 par 2. Avec une autre implémentation il serait donc possible que l'on voit une amélioration des performances avec l'algorithme glouton.

4.3 Amélioration du branchement

Pour tester l'efficacité des nouvelles méthodes de branchement on va effectuer cette expérience :

- Générer n graphes différents ordonnés par leur nombres de sommets et d'arêtes de manière croissante.
- Pour chaque graphe :
  - S'assurer que les différentes méthodes de branchement retournent bien une couverture.
  - Comparer le nombre de nœuds exploré pour les différentes méthodes.

On obtient ces résultats :

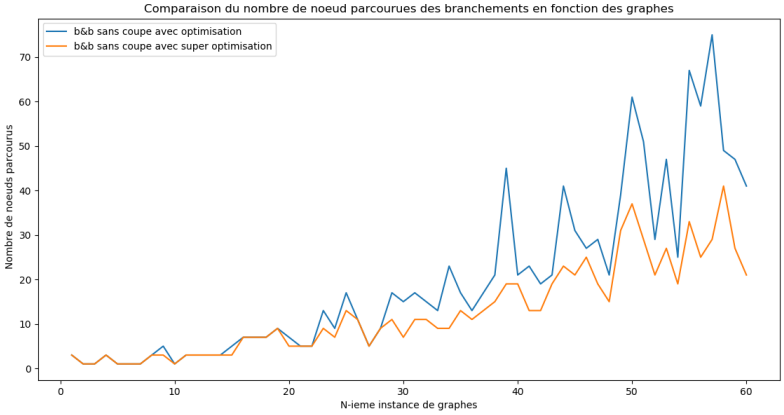

FIGURE 8 – Comparaison des optimisations de B&B

On peut remarquer que l'optimisation du branchement réduit encore le nombre de nœuds parcourus. En effet cela est dû au fait qu'il n'y a plus de doublon au niveau des branchements d'arêtes lors de la construction de l'arbre. En améliorant le branchement on a ainsi réduit le nombre de nœuds que l'on parcourra au maximum.

Supposons un graphe G connexe d'au moins 3 sommets et u, un sommet de G de degré 1.
Montrons qu'il existe toujours une couverture optimale de G qui ne contient pas u :

Supposons par l'absurde qu'il n'existe pas de couverture optimale de G qui ne contient pas u. Cela voudrait dire que le sommet voisin Ng(u) n'offre pas de couverture équivalente ou plus du sommet u. Or cela est impossible, car le sommet Ng(u) couvre la même arrête que u. De plus, G est connexe, Ng(u) est au moins de degré 2, donc Ng(u) peut aussi couvrir d'autres arêtes. Donc c[u] $\subseteq$ c[Ng(u)]

L'intérêt d'ajouter ce test est de supprimer toutes les couvertures dites "inutiles". Ainsi, lors de la construction de l'arbre, moins d'étapes seront nécessaires pour atteindre les feuilles.

4.4 Qualité des algorithmes approchés

L'implémentation de l'algorithme de B&B avec une coupe et un ε peut être considéré comme une heuristique qui a été implémenté. On aurait aussi pu avoir un algorithme qui permet de choisir aléatoirement un sommet de degré ≥ 1 dans le graphe pour le mettre dans la couverture, mais ça ne s'est pas fait par manque de temps. De même pour l'approximation des algorithmes couplage et glouton.

4.5 Conclusion

Après l'implémentation de ces différents algorithmes, nous avons pu observer diverses façon de trouver une couverture d'arêtes d'un graphe. Nous avons rencontré des difficultés de solutions qui n'étaient pas optimales et de temps d'exécution exponentiels.
Cela dit, on a réussi à trouver des manières d'optimiser ces algorithmes et les mettre en lien pour obtenir un résultat optimal et assez rapide grâce au B&B et ses améliorations ainsi qu'à l'ajout de bornes.
Donc, quand bien même la couverture d'arêtes est NP-Difficile, nous voyons avec ce problème que nous pouvons tout de même gagner en efficacité de différentes façons.