PROJET COMPLEX MU4IN900

RAPPORT

Couverture de Graphe

Étudiants :
Anyes TAFOUGHALT
Sama SATARIYAN

Encadré par :
Fanny PASCUAL

21 octobre 2023

Table des matières

1 Introduction

Ce projet aborde le problème de Vertex Cover dans le domaine des graphes non orientés, cherchant à déterminer une couverture minimale pour le graphe, où chaque arête doit avoir au moins un de ses sommets inclus dans cette couverture. Dans les sections suivantes, nous explorerons les algorithmes exactes et approchés pour aborder ce problème NP-difficile et partagerons les résultats issus des tests expérimentaux menés sur diverses instances de graphes.

2 Graphes

Pour ce projet, nous avons choisi de mettre en œuvre nos algorithmes en utilisant le langage de programmation Python. Cette décision a été motivée par l'utilisation de la bibliothèque NetworkX, qui a grandement simplifié la création des fonctions fondamentalles et la représentation visuelle des graphes.

2.1 Opérations de base

L'utilisation de la bibliothèque NetworkX a simplifié la mise en place des fonctionnalités de base. Par exemple, pour créer un graphe vide, nous avons employé la fonction Graph(). L'ajout des nœuds et des arêtes est une opération intuitive grâce aux méthodes add_node() et add_edge(), tandis que pour récupérer des informations, telles que le degré d'un nœud, il nous a suffi de consulter G.degree[node], où G représente le graphe. En outre, pour supprimer un nœud du graphe, nous avons fait appel à remove_node(node).

2.2 Génération d'instances

Pour générer des instances aléatoires, nous avons utilisé la fonction de la bibliothéque NetworkX **fast_gnp_random_graph(n,p)**. Cette méthode permet de créer efficacement un graphe aléatoire composé de n nœuds, où la probabilité d'avoir une arête entre chaque paire de nœuds est définie par p. Une valeur élevée de p conduit à un graphe plus dense.

3 Méthodes approchées

3.1 Analyse de la Performance de l'Algorithme Glouton

Après avoir implémenté les deux algorithmes de couplage et de glouton, nous allons d'abord démontrer que l'algorithme glouton n'aboutit pas toujours à une solution optimale. Pour ce faire, nous utiliserons un contre-exemple illustratif.

Considérons le graphe suivant :

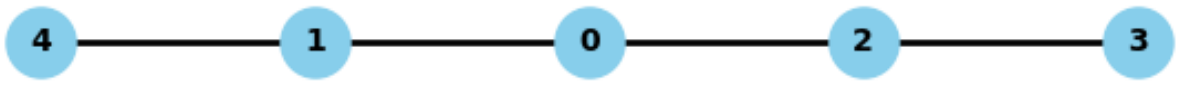

FIGURE 1 – Graphe utilisé pour le contre exemple

L'algorithme glouton retourne la solution [0, 1, 2], qui a une taille de 3. Cependant, la solution optimale pour ce graphe est [1, 2], qui a une taille de 2. Cette différence de taille montre clairement que l'algorithme glouton ne trouve pas la solution de taille minimale dans cet exemple spécifique.

Pour poursuivre la démonstration et montrer que l'algorithme glouton n'est pas r-approché pour un certain r en utilisant l'exemple précédent, nous allons prendre $r < \frac{3}{2}$.

Supposons, par l'absurde, que l'algorithme glouton est r-approché, où $r < \frac{3}{2}$. Cela signifierait que la taille de la solution gloutonne doit être d'au plus r fois la taille de la solution optimale. Or dans notre exemple, la taille de la solution gloutonne est de 3, tandis que la solution optimale a une taille de 2 et $3 > r \cdot \text{OPT}$ avec $\text{OPT} = 2$. Par conséquent, l'algorithme glouton n'est pas r-approché pour $r < \frac{3}{2}$.

3.2 Comparaison des Performances entre l'Algorithme de Couplage et l'Algorithme Glouton

Pour établir une comparaison approfondie et précise entre les deux algorithmes, nous avons mis en place une procédure systématique. Cette procédure implique l'exécution des algorithmes sur une série de graphes générés aléatoirement, en faisant varier plusieurs paramètres. Nous avons exploré différentes valeurs de probabilité p, notamment 0.25, 0.50, 0.75, 1, et ajusté la taille des graphes de manière progressive, allant de nMin=0 à nMax=400 avec un pas de nMax/ n_iter = 50 à chaque itération.

Cette démarche nous a permis de comparer les performances des deux algorithmes en termes de temps d'exécution et de la taille de la couverture qu'ils retournent. Pour garantir la fiabilité des résultats, nous avons répété l'exécution des algorithmes sur 5 graphes différents pour chaque combinaison de taille de graphe et de valeur de p, puis avons calculé la moyenne des performances obtenues.

Comparons à présent les performances de l'algorithme glouton et de l'algorithme de couplage.

Tout d'abord, il est essentiel de noter que, d'aprés la figure 2, l'algorithme glouton se révèle notablement plus lent que l'algorithme de couplage. Cette observation reste valable pour l'ensemble des graphes, indépendamment de la valeur de "p". En effet, lorsque "p" augmente, entraînant une augmentation de la densité du graphe, la complexité de l'algorithme glouton augmente significativement. La raison en est que lorsque la densité d'un graphe augmente, l'algorithme glouton nécessite plus de temps pour explorer l'ensemble des arêtes en quête des nœuds présentant le degré maximum. Cela s'applique également au nombre de sommets ; à mesure que le nombre de sommets dans le graphe augmente, le temps d'exécution de l'algorithme augmente, car cela implique de parcourir un nombre croissant de nœuds pour identifier la couverture.

En examinant les graphes de la figure 3 comparant la taille de la couverture obtenue par chaque algorithme, nous constatons que l'algorithme glouton semble être légèrement plus efficace que l'algorithme de couplage lorsque les graphes sont peu denses. Toutefois, cette différence tend à s'estomper à mesure que la densité du graphe augmente.

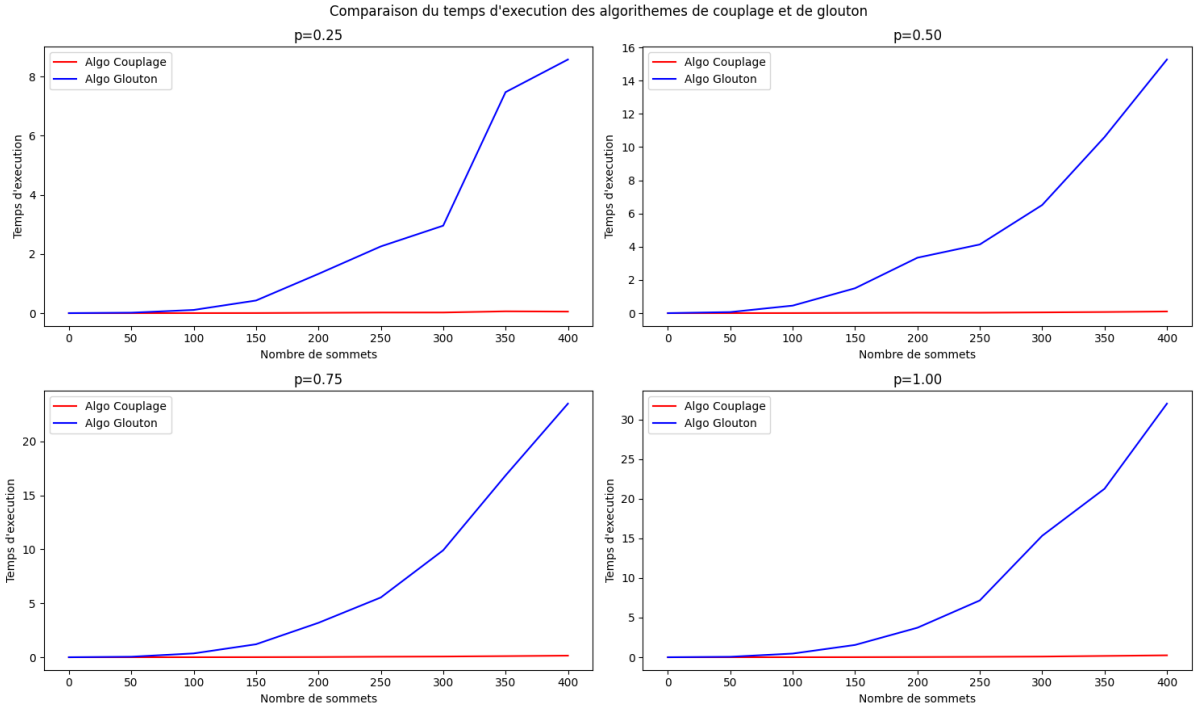

FIGURE 2 – Comparaison du temps d'execution entre l'algorithme de couplage et de glouton

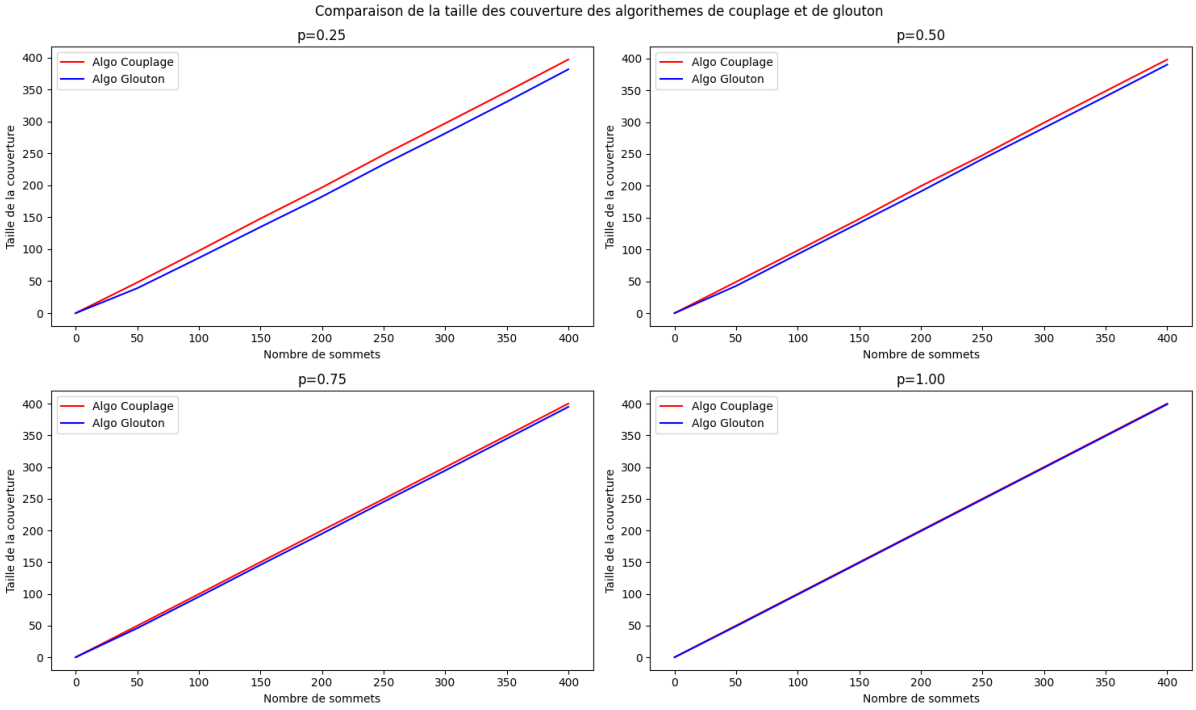

FIGURE 3 – Comparaison de la taille des couverture entre l'algorithme de couplage et de glouton

4 Séparation et évaluation

Dans cette section nous allons présenter les différents algorithmes de Brand And Bound que nous avons réussi à implémenter afin de trouver la solution optimale en essayant d'optimiser à chaque fois le nombre de noeuds à visiter.

4.1 Branchement

En premier lieu nous avons implémenté l'algorithme de branchement qui explore de manière systématique toutes les combinaisons possibles de sommets dans un graphe afin de trouver une couverture minimale. Il commence par initialiser une pile contenant le graphe d'origine et une couverture vide. Ensuite, il itère en extrayant un couple (graphe, couverture) de la pile. Si le graphe n'a plus d'arêtes, cela signifie qu'une nouvelle couverture a été trouvée, et si elle est plus petite que la meilleure couverture trouvée jusqu'à présent, elle est mise à jour. Sinon, l'algorithme choisit une arrte du graphe et le divise en deux sous-graphes en supprimant une extrémité de l'arrête dans chaque sous graphe et continue à explorer ces deux branches. L'algorithme se termine en renvoyant la meilleure couverture trouvée.

Dans le but d'évaluer les performances de cet algorithme, nous avons examiné le temps d'exécution ainsi que le nombre de noeuds visités en utilisant des graphes aléatoires. Ces graphes sont générés avec un nombre de sommets n et une probabilité p.

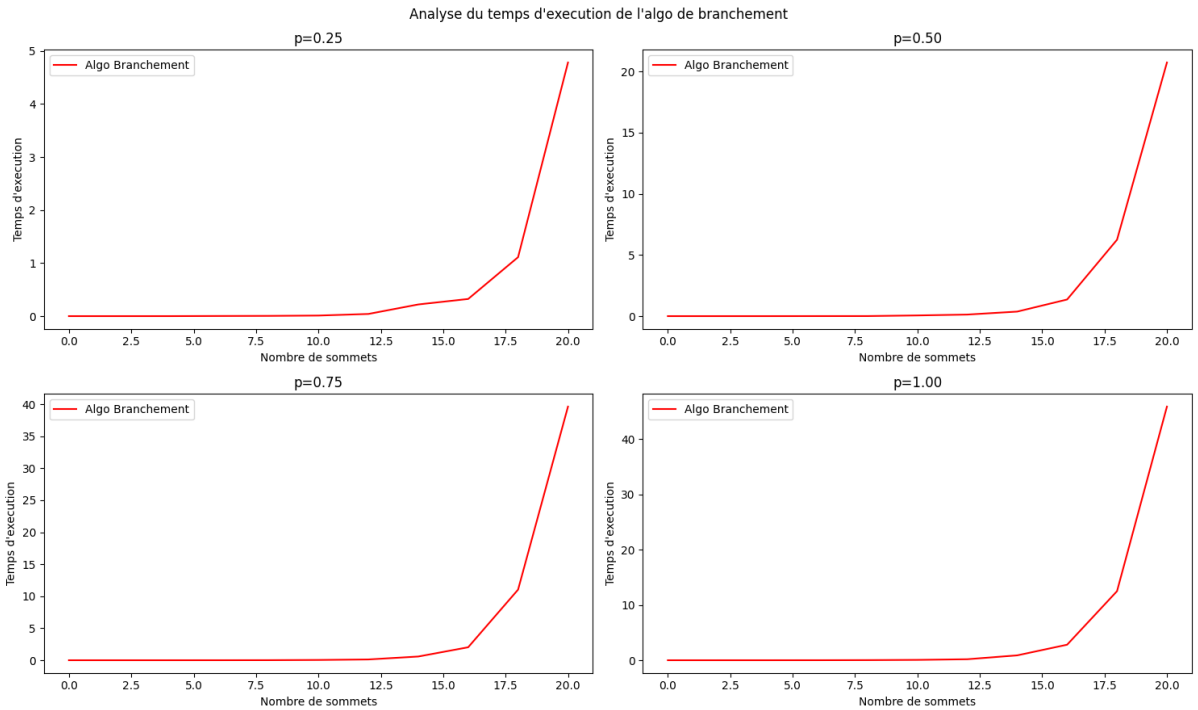

FIGURE 4 – Analyse du temps d'execution de l'algorithme de branchement

Suite à l'analyse des graphes des deux figures 4 et 6, il est évident que l'algorithme que nous avons mis en œuvre nécessite plus de temps d'exécution à mesure que le nombre de sommets augmente et lorsque la probabilité "p" augmente également. De plus, le nombre de nœuds visités augmente considérablement, expliquant ainsi les longs temps

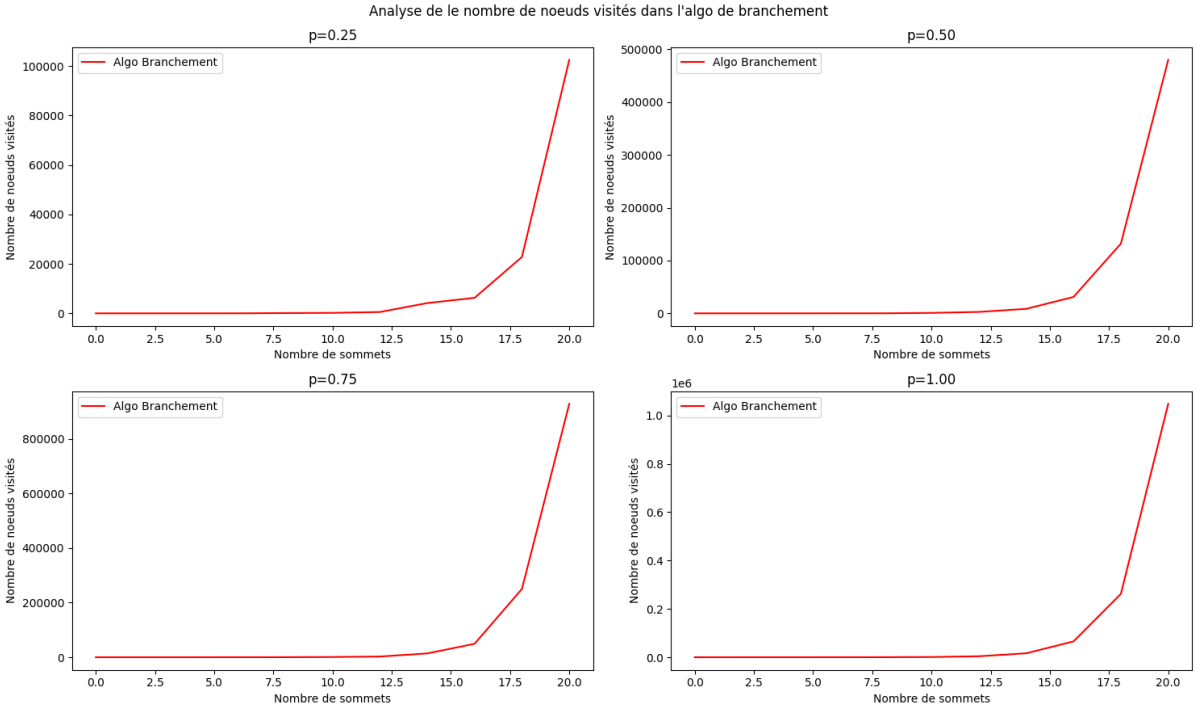

FIGURE 5 – Analyse du nombre de noeuds visités dans l'algorithme de branchement

d'exécution. Cela est dû au fait que notre algorithme explore l'ensemble des nœuds de l'arbre de recherche, testant ainsi toutes les combinaisons possibles de couvertures pour trouver la plus optimale.

Afin d'améliorer l'efficacité de cet algorithme, nous envisageons d'introduire des bornes, permettant ainsi d'élaguer les nœuds de l'arbre de recherche qui mènent à des solutions moins optimales par rapport à celles déjà trouvées. Cette optimisation réduira la taille des nœuds visités et contribuera à accélérer l'algorithme.

4.2 Ajout de bornes

L'objectif de cette section est d'optimiser l'algorithme précédent en lui ajoutant des bornes. Soit G un graphe, M un couplage de G et C une couverture de G. Alors :

$$|C| \geq \max\{b_1, b_2, b_3\}$$

Avec b1 $= \left\lfloor \frac{m}{\Delta} \right\rfloor$ où Δ est le degré maximum des sommets du graphe, b2 $= |M|$, b3 $= \frac{2n-1-\sqrt{(2n-1)^2-8m}}{2}$

Montrons la validité des 3 bornes :

- **La borne b1 :**

  La taille de la couverture |C| doit être supérieure ou égale au quotient de m (le nombre d'arêtes du graphe) divisé par Δ (le degré maximum des sommets du graphe). Cette borne repose sur le fait qu'il faut au moins un sommet dans la couverture pour couvrir chaque arête.

  Si m est le nombre total d'arêtes et Δ est le degré maximum des sommets, alors la division entière de m par Δ, qui représente un nombre de sommets suffisant pour

couvrir les arêtes. En d'autres termes, si vous avez m arêtes et que chaque sommet a un degré maximal de Δ, alors il faut $\frac{m}{\Delta}$ sommets pour couvrir ces m arêtes. Donc, cette borne dit que la taille de la couverture doit être au moins $\lfloor \frac{m}{\Delta} \rfloor$.

- **La borne b2 :**

  Pour montrer que la taille d'une couverture est toujours supérieure ou égale au nombre d'arêtes d'un couplage M, nous allons utiliser un argument simple basé sur la nature des couplages et des couvertures. Un couplage M est un sous-ensemble d'arêtes du graphe dans lequel aucune paire d'arêtes partage un sommet commun. Par conséquent, pour couvrir toutes les arêtes de ce couplage, il faut au moins un sommet distinct pour chaque arête du couplage.

  Donc, |C| doit contenir au moins autant de sommets que |M| pour couvrir toutes les arêtes du couplage. Formellement, nous pouvons écrire : $|C| \geq |M|$.

  Cela signifie que la taille d'une couverture est toujours supérieure ou égale au nombre d'arêtes d'un couplage.

- **La borne b3 :**

  Montrons que $|C| \geq \frac{2n-1-\sqrt{(2n-1)^2-8m}}{2}$.

  On sait que si un graphe avec n sommets est complet, alors son nombre d'arêtes est $m = \frac{n(n-1)}{2}$. Par conséquent, pour tout graphe, on a $m \leq \frac{(n-1)n}{2}$.

  Soit $f(x) = x^2 + (1-2n)x + 2m$. En substituant $x = |C|$, nous avons $f(|C|) = |C|^2 + (1-2n)|C| + 2m$.

  Comme $|C| \leq n$, nous avons $f(|C|) \leq n^2 + (1-2n)n + 2m$, ce qui simplifie à $n^2 + (1-2n)n + 2m \leq 1 - n$.

  Pour tout graphe, $n \geq 0$, ce qui implique $f(|C|) \leq 0$.

  Trouvons les valeurs de $|C|$ pour lesquelles $f(|C|) \leq 0$:

  Calculons le discriminant : $\Delta = (1-2n)^2 - 8m = 4(n^2 - n - 2m) + 1$.

  Montrons que $n^2 - n - 2m \geq 0$: On sait que $m \leq \frac{(n-1)n}{2}$, donc $2m \leq \frac{(n-1)n}{2}$, ce qui simplifie à $0 \leq n^2 - n - 2m$.

  Par conséquent, $f(|C|) = 0$ admet deux solutions x_1 et x_2 :

  $x_1 = -\frac{(1-2n)-\sqrt{(1-2n)^2-8m}}{2}$

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

  Ainsi, $f(|C|) \leq 0$ si et seulement si $x_1 \leq |C| \leq x_2$. Cela se traduit par

  $-\frac{(1-2n)-\sqrt{(1-2n)^2-8m}}{2} \leq |C| \leq -\frac{(1-2n)+\sqrt{(1-2n)^2-8m}}{2}$.

  En simplifiant, nous obtenons $\frac{2n-1-\sqrt{(2n-1)^2-8m}}{2} \leq |C|$.

L'algorithme **branch and bound** est une méthode utilisée pour résoudre des problèmes d'optimisation combinatoire. Il opère en calculant des bornes pour guider la recherche. A fin de mettre ça en place, tout d'abord, nous avons implémenté la fonction bornes(G, M) qui calcule trois bornes : b1, b2 et b3. Ensuite, la fonction branchement_bornes(Graph) met en œuvre l'algorithme **branch and bound** pour rechercher une couverture minimale dans le graphe initial. Elle commence par initialiser une pile avec le graphe initial et une liste vide pour stocker la couverture actuelle. Ensuite, elle calcule une couverture approximative à l'aide de l'algorithme de couplage et initialise une borne supérieure (Bsup) avec la taille de cette couverture. L'algorithme explore ensuite

les branches de l'arbre de recherche en supprimant séquentiellement les nœuds du graphe. S'il parvient à une branche avec un sous-graphe sans arêtes, il met à jour Bsup s'il trouve une couverture plus petite. Sinon si la borne inférieure (Binf) qui est égale au max de b1, b2 et b3 d'une branche est supérieure à Bsup, l'exploration de cette branche est abandonnée. Sinon, les deux branches sont explorées en supprimant une extrimité d'une arrête du graphe d'origine. Finalement, l'algorithme renvoie la meilleure couverture trouvée avec moins de noeuds visités par rapport à l'algorithme précédent du branchement.

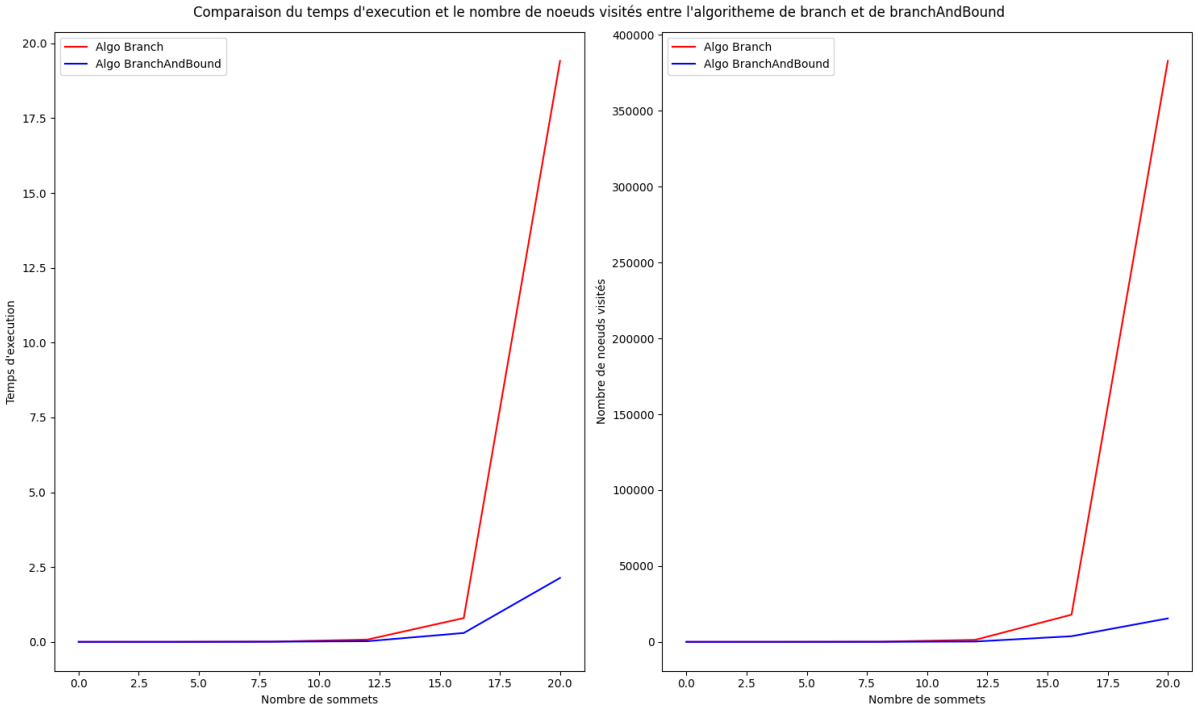

FIGURE 6 – Comparaison du temps d'execution et le nombre de noeuds visités entre l'algorithme de branchement et de branchAndBound

Nous avons ensuite comparer le temps d'execution des 2 algorithmes pour conclure que branch and bound et plus rapide et visite moins de noeuds par rapport à l'algorithme de branchement.

4.3 Amélioration du branchement

Dans cette section, nous allons évaluer les performances de deux améliorations de l'algorithme Branch and Bound.

La première amélioration consiste à diviser les branches de manière à optimiser le choix des sommets. Lorsque nous divisons sur une arête u, v, dans la deuxième branche où nous incluons le sommet v dans la couverture, nous supposons que le sommet u n'est pas inclus (le cas où il est inclus étant traité dans la première branche). Dans cette deuxième branche, en n'incluant pas u dans la couverture, nous devons alors ajouter tous les voisins de u, que nous pouvons également supprimer du graphe. Cette approche vise à minimiser le nombre de sommets inclus dans la deuxième branche.

La deuxième amélioration réside dans le choix de la manière dont nous branchons. L'idée est de choisir la branche de manière à ce que le sommet u soit de degré maximum

dans le graphe restant. Cette approche vise à éliminer un maximum de sommets dans la deuxième branche.

Nous allons mettre ces nouvelles méthodes à l'épreuve et analyser les résultats obtenus pour évaluer leur efficacité par rapport à l'algorithme de branch and bound de base.

En examinant le Graphique 7, nous pouvons constater que les modifications apportées aux algorithmes se traduisent par une réduction significative du temps d'exécution et du nombre de nœuds visités.

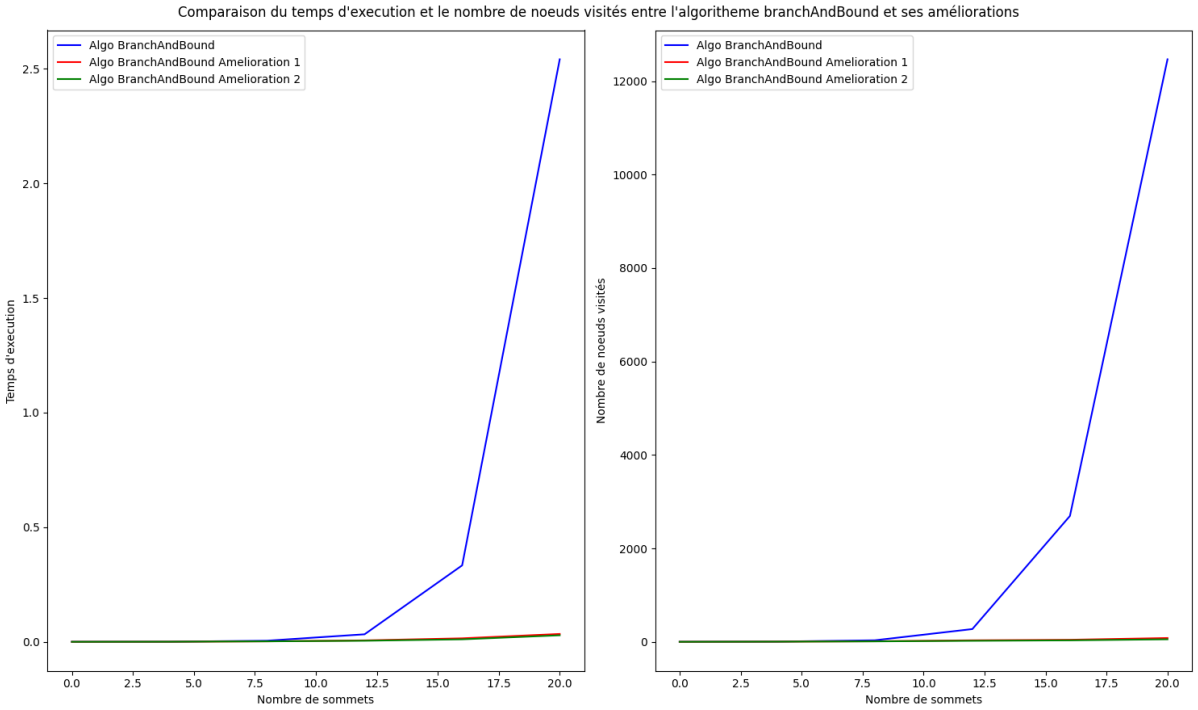

FIGURE 7 – Comparaison du temps d'execution et le nombre de noeuds visités entre l'algorithme de branchAndBound et ses améliorations

En observant le Graphique 8, il est clair que la deuxième amélioration de l'algorithme surpasse la première en termes de performances, car elle nécessite la visite d'un nombre nettement inférieur de nœuds. Montrons que dans un graphe G, si un sommet u est de degré 1 alors il existe toujours une couverture optimale qui ne contient pas u.

Considérons un graphe G contenant l'arête u, v, où u est un sommet de degré 1 dans G. Il est évident que pour toute couverture optimale de G, au moins l'un des sommets de l'arête u, v doit en faire partie. Le sommet u couvre uniquement l'arête u, v, tandis que le sommet v couvre au moins cette arête ainsi que potentiellement d'autres. Supposons que nous ayons une couverture optimale C pour G, et supposons que C contienne le sommet u. Dans ce cas, nous pouvons remplacer u par v dans C sans changer la taille de la couverture, et nous pouvons être certains que C reste une couverture optimale pour G. En effet, la taille de C reste inchangée, et toutes les arêtes de G sont toujours couvertes

L'ajout de cette vérification vise à éliminer les couvertures considérées comme "inutiles". Par conséquent, lors de la construction de l'arbre, cela réduit le nombre d'étapes nécessaires pour atteindre les feuilles.

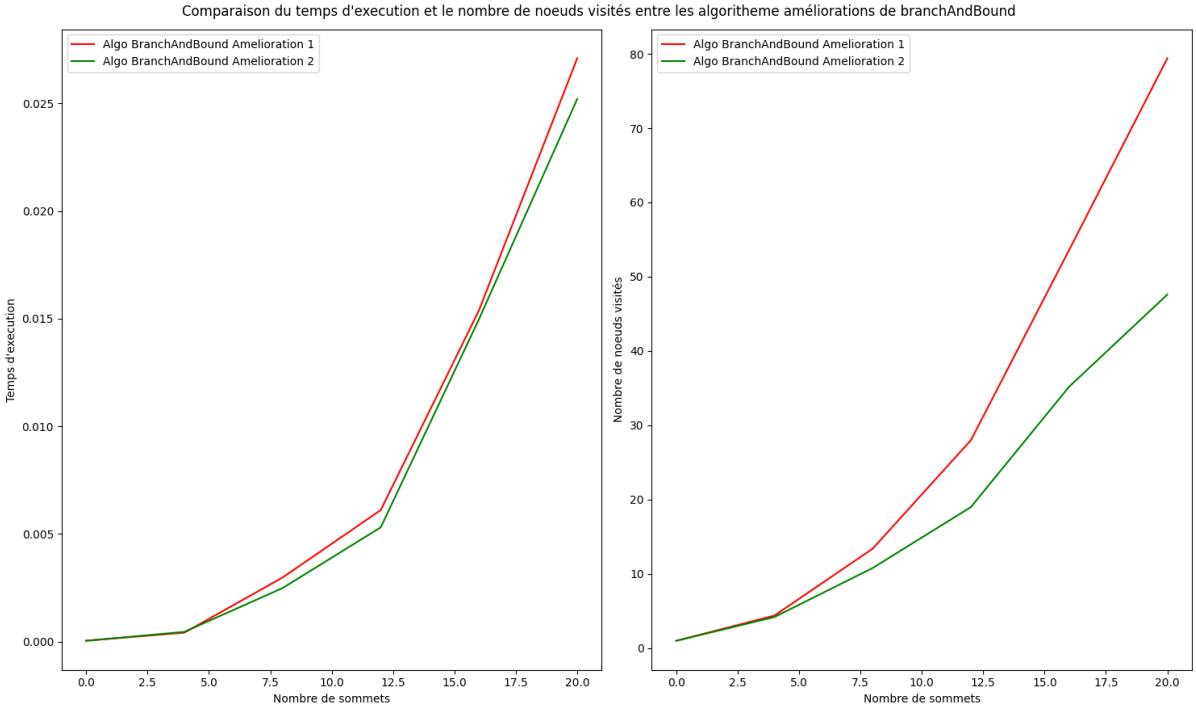

FIGURE 8 – Comparaison du temps d'execution et le nombre de noeuds visités entre les algoritheme améliorations de branchAndBound

4.4 Qualité des algorithmes approchés

Selon le graphique de la figure 9, il est évident que l'algorithme glouton offre un rapport d'approximation systématiquement inférieur à celui de l'algorithme de couplage. Cette observation suggère que l'algorithme glouton fournit des solutions approximatives plus proches de l'optimal. En d'autres termes, il est plus fiable car il se rapproche davantage de la solution optimale.

Les pires rapports d'approximation obtenus pour les deux algorithmes sont les suivants :

- Algorithme glouton : 1.1
- Algorithme de couplage : 2

Cela signifie que le rapport d'approximation le plus élevé atteint par l'algorithme glouton est de 1.1, tandis que l'algorithme de couplage atteint un rapport de 2, ce qui confirme que l'algorithme glouton fournit de meilleures solutions approximatives dans l'ensemble.

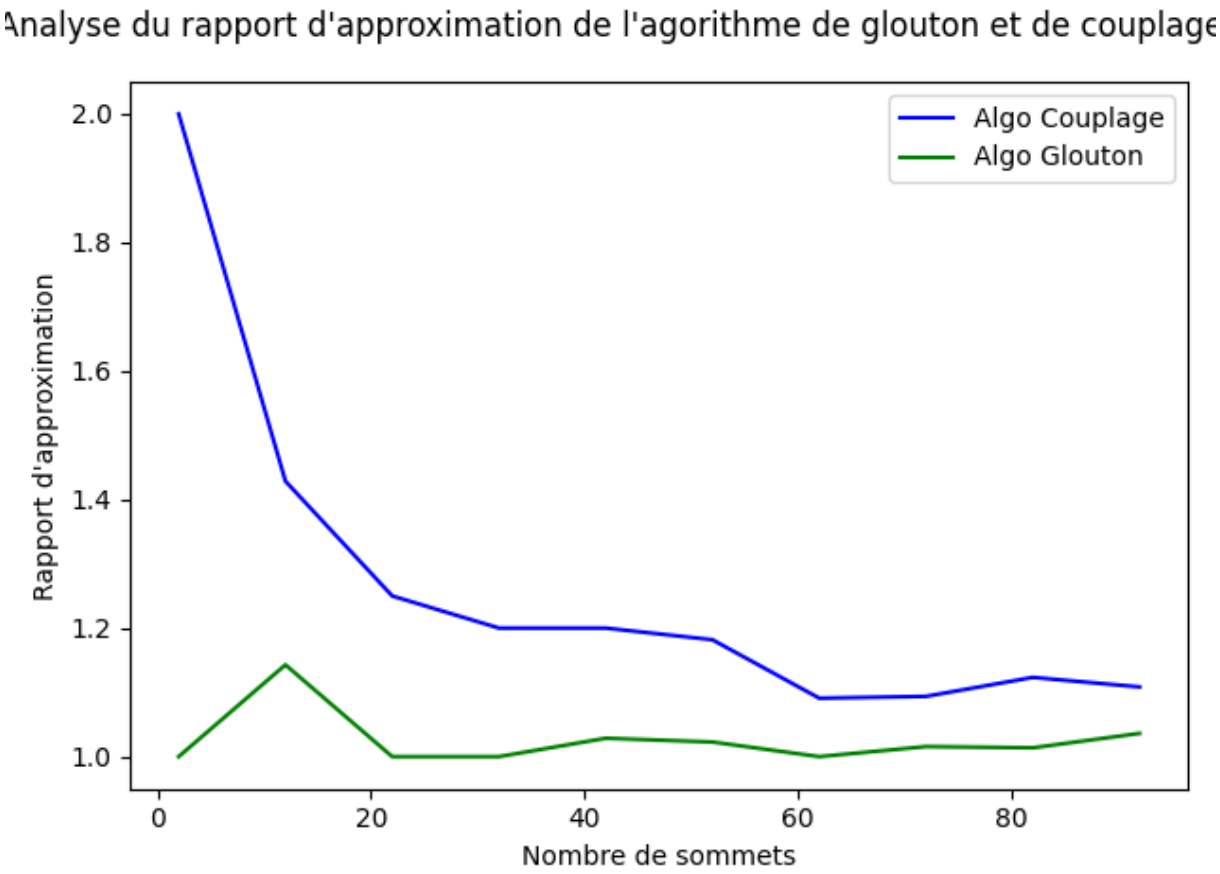

FIGURE 9 – Analyse du rapport d'approximation de l'agorithme de glouton et de couplage