PROJET COMPLEXITÉ, ALGORITHMES RANDOMISÉS ET APPROCHÉS

# COUVERTURE DE GRAPHE

Assia MASTOR
28602563
Amel MOKDADI
21309654

MASTER CCA
2023-2024

Enseignant
Bruno Escoffier

SORBONNE UNIVERSITÉ

# SOMMAIRE

# Introduction

Tout au long de ce projet, nous allons traiter le problème VERTEX COVER qui consiste à trouver une couverture d'un graphe G = (V,E) non orienté , avec V l'ensemble des n sommets et E l'ensemble des m arêtes de G contenant un nombre minimal de sommets. Nous allons ainsi implémenter différentes fonctions afin de comparer leur efficacité. Ces différents algorithmes pourront être exacts ou approchés.

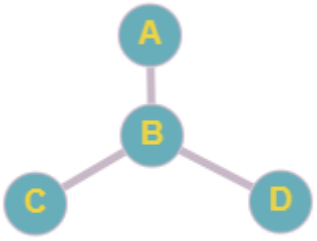
# Graphes

Pour ce projet, nous avons choisi d'utiliser le langage de programmation Python pour mettre en pratique les méthodes requises. Dans notre approche conceptuelle, un graphe est symbolisé à l'aide d'une structure de données appelée dictionnaire, notée $Dict = \{ k : val \}$, où :

- Chaque nœud du graphe est représenté par une clé $k$.

- Chaque clé $k$ est associée à une valeur $val$ qui est une liste contenant les nœuds voisins du nœud correspondant.

Pour illustrer ceci, prenons un exemple de graphe $G = (V, E)$, où $V$ = { A, B, C, D } et $E$ = { (A, B), (B, C), (B, D) }.




Dans ce cas, le dictionnaire qui représente le graphe $G$ sera établi de la manière suivante :

$G$ ={ 'A': ['B'],' B': ['A',' C',' D'],' C': ['B'],' D': ['B']}.

Pour commencer, voici 2 tests des fonctions supprsommet et supprsommets prenant G et un ou plusieurs sommets à supprimer et renvoyant le graphe après suppression.

```
[8]: import projet
```

```
[9]: G = {
         'A': ['B', 'C'],
         'B': ['A', 'C'],
         'C': ['A', 'B']
     }
```

```
[10]: print(projet.supprsommet(G, 'A'))
```

```
{'B': ['C'], 'C': ['B']}
```

```
[11]: G = {
          'A': ['B', 'C'],
          'B': ['A', 'C'],
          'C': ['A', 'B']
      }
```

```
[12]: print ( projet.supprsommets(G, ['A', 'B']))
```

```
{'C': []}
```

Également ,voici deux seconds tests des fonctions permettant de savoir le nombre de degré de chacun des sommets du graphe.

```
[15]: print(projet.degres(G))
```

```
{'A': 2, 'B': 2, 'C': 2}
```

```
[16]: G = {
          'A': ['B', 'C'],
          'B': ['A', 'C'],
          'C': ['A', 'B']
      }
```

```
[17]: print(projet.degresmax(G))
```

```
A
```

Nous avons ensuite la fonction Aleatoire(n,p) qui nous permet d'obtenir un graphe aléatoire de n sommets ayant une p probabilité d'avoir les sommets i et j.

```
print(projet.aleatoire(5,0.5))
```

```
{'D': ['D', 'D', 'Q', 'E', 'Q'], 'R': ['Q'], 'E': ['D', 'Q'], 'Q': ['D
', 'D', 'R', 'E']}
```

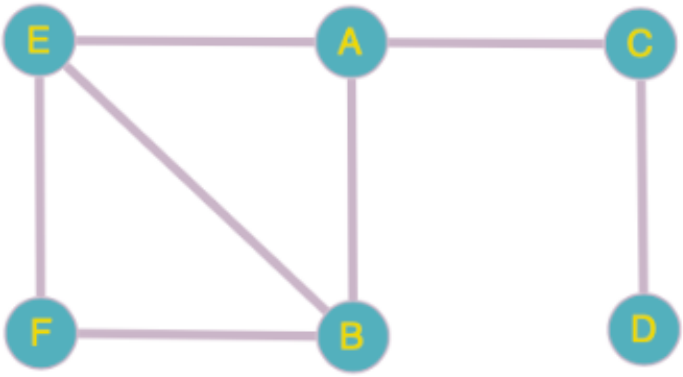
# Méthodes approchées

L'algorithme glouton fonctionne en sélectionnant, à chaque étape, le sommet ayant le plus grand nombre de connexions, dit de degré maximum, dans le graphe pour l'ajouter à la couverture. Cette approche a pour conséquence de minimiser le nombre d'arêtes restantes

dans le graphe le plus efficacement possible à chaque itération, réduisant ainsi rapidement le nombre total d'arêtes dans le graphe. Cependant, il ne renvoie pas forcément la solution optimale.

On peut facilement voir dans l'exemple ci dessous que la valeur de retour de l'algorithme glouton n'est pas la couverture optimale.

En effet, la couverture optimale du graphe 1 est {B, C, E} de cardinalité 3 tandis que l'algorithme nous renvoie une solution de cardinalité 4. Il ne nous renvoie donc pas la solution optimale, il n'est donc pas optimal.




Graphe 1

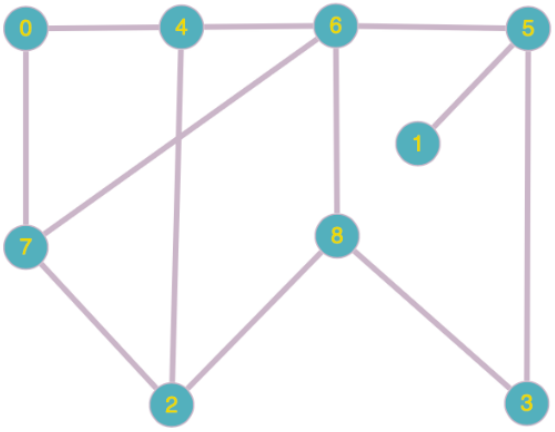
```
G = {
    'A': ['E','B', 'C'],
    'B': ['A', 'E', 'F'],
    'C': ['A', 'D'],
    'D': [ 'C'],
    'E': ['A','B','F'],
    'F': ['E','B']
}


print(projet.algo_glouton(G))
```

```
{'C', 'A', 'E', 'B'}
```

L'algorithme glouton est un algorithme qui n'est pas r-approché. Trouvons à présent le r. On commence par calculer le rapport d'approximation avec cet exemple.

On a r = $\frac{|C|}{|C_{optimal}|} = \frac{4}{3} \simeq 1{,}333 > 1.2$.

Nous pouvons également donner un second exemple comme le graphe suivant:




Graphe 2

```
G = {
      0: [4, 7],
      1: [5],
      2: [7,4,8],
      3: [8,5],
      4: [0,2,6],
      5: [6,1,3],
      6: [4,8,5],
      7: [0,2],
      8: [2,3,6]
}
print(projet.algo_glouton(G))
```

```
{0, 2, 3, 5, 6}
```

L'algorithme nous renvoie une solution de cardinalité 5 tandis que la solution optimale , à savoir {4,5,8,7} est de cardinalité 4. En appliquant la formule du rapport d'approximation on

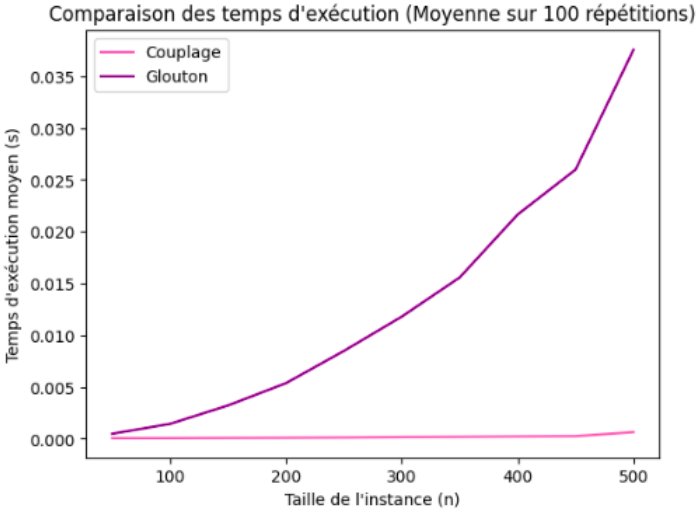
a r = $\frac{|C|}{|C_{optimal}|} = \frac{5}{4} = 1.25 > 1.2$. On peut donc en déduire que l'algorithme glouton est un algorithme qui n'est pas r-approché avec r = 1.2.

L'algorithme de couplage quant à lui, construit une couverture en ajoutant les extrémités de chaque arête à la couverture, à condition qu'aucune de ces extrémités ne soit déjà présente dans la couverture. Cette étape est répétée jusqu'à ce que tout le graphe soit parcouru. L'algorithme retourne une couverture qui n'est pas nécessairement optimale, sous forme d'une liste de sommets formant le couplage.

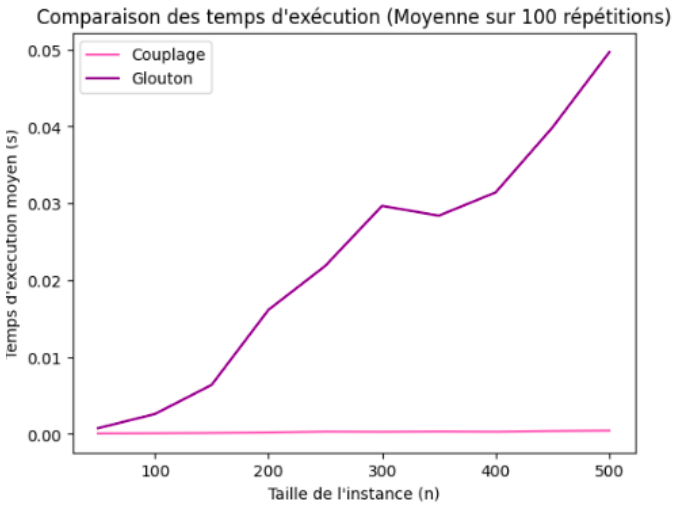
Nous allons maintenant comparer les algorithmes glouton et de couplage du point de vue de leur temps de calcul et de la qualité des solutions retournées. Pour cela, nous avons créé une fonction de comparaison qui génère des graphes aléatoires de différentes tailles, en augmentant la taille du graphe par pas de n/10. Pour chacune des tailles, on exécute l'opération 100 fois, à savoir, calculer le temps d'exécution de l'algorithme de couplage et de l'algorithme glouton. On effectue ensuite une moyenne et génère un graphique comparatif.

Comparaison du point de vue de temps d'exécution de la solution :

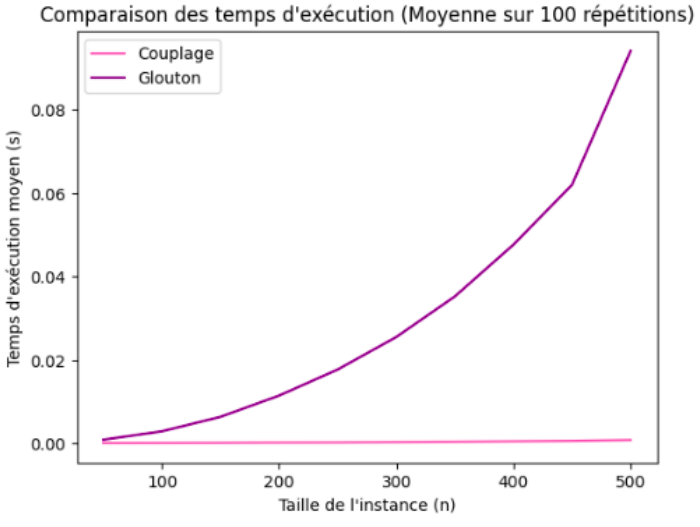
p = 0.25




p = 0.5




p = 0.75



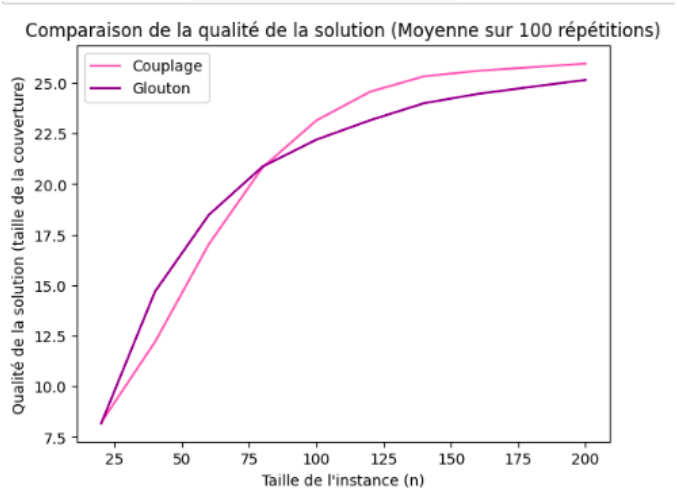

Nous pouvons voir que l'algorithme couplage est nettement plus rapide que l'algorithme glouton, qui voit son temps d'exécution augmenter en fonction du nombre de sommets du graphe. Aussi, on remarque que plus la probabilité p d'avoir une arête entre 2 sommets augmente, plus le temps d'exécution de glouton augmente.

L'algorithme glouton à une complexité de O($n^2$). En effet, dans le pire cas, il parcourt pour chaque sommet, tous les autres sommets pour vérifier les arêtes.

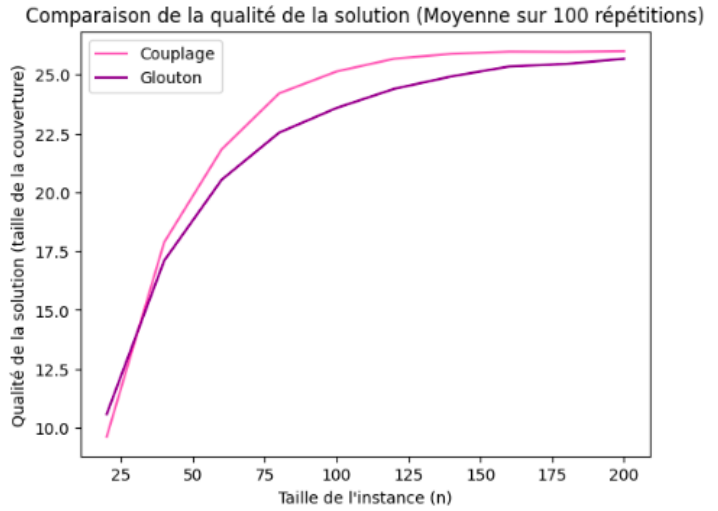
L'algorithme couplage, lui, parcours chacune des arêtes puis ses voisins restants, il a donc une complexité inférieur à O($n^2$). Cela explique donc pourquoi l'algorithme glouton est bien plus lent que couplage.

Comparaison du point de vue de la qualité de la solution :

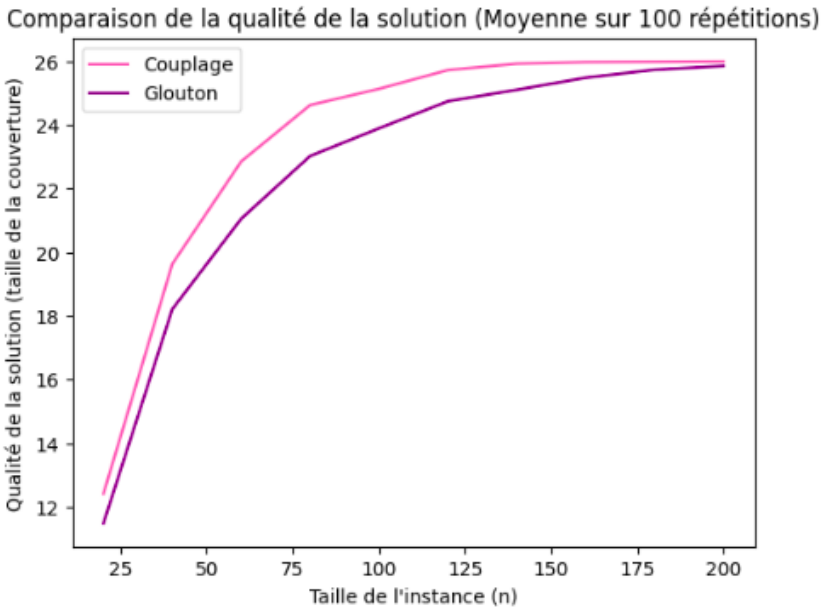
p = 0.2




p = 0.5




p = 0.75



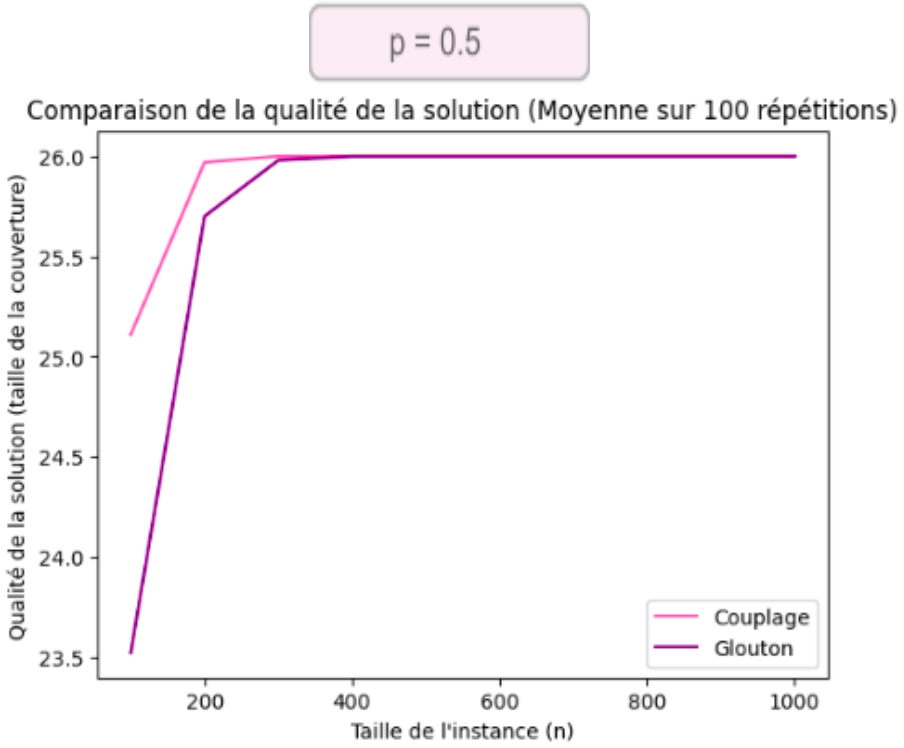

Le critère principal pour évaluer la qualité des solutions consiste à prendre en compte la dimension, c'est-à-dire la longueur de la couverture. Ainsi, nous comparons la dimension des solutions fournies par les deux algorithmes.

Pour effectuer les tests, nous avons effectué 100 essais pour 200 valeurs différentes de n.

Nous avons posé Nmax à environ 2000. Au-delà de 2000 sommets l'algorithme glouton met de nombreuses secondes à s'exécuter. On constate que glouton renvoie généralement des couvertures de tailles inférieures à celles de couplage. On en déduit donc que, en termes de qualité, l'algorithme glouton se révèle supérieur, même s'il nécessite un

temps d'exécution plus long. Il se rapproche plus de la solution optimale. On va donc préférer Glouton pour trouver une couverture en fonction du n.

Lorsqu'il y a un grand nombre d'arêtes, la différence peut être considérée comme négligeable comme on peut le voir sur ce graphique comparatif:



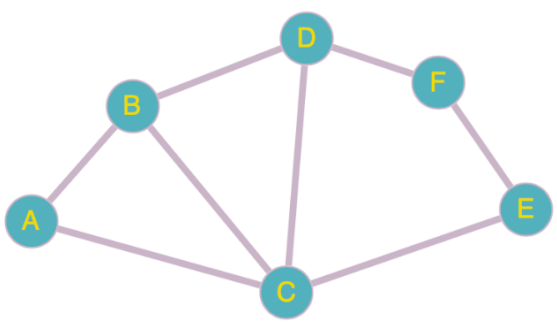

# Séparation et évaluation

## Branchement

Dans cette partie, nous implémentons différents algorithmes de branchement. Voici un exemple d'exécution de la fonction branchement avec le graphe G1. Il retourne bel et bien la solution optimale.

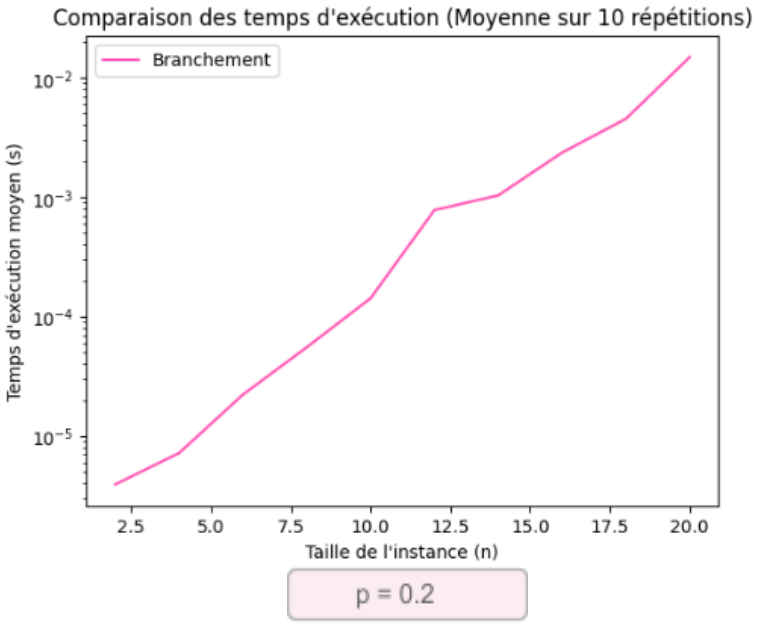
```
G1 = {
    'A': ['B', 'C'],
    'B': ['A', 'C', 'D'],
    'C': ['A', 'B', 'D', 'E'],
    'D': ['B', 'C', 'F'],
    'E': ['C', 'F'],
    'F': ['D', 'E']
}

result = projet.branchement(G1)
print("Meilleure solution:", result)
```

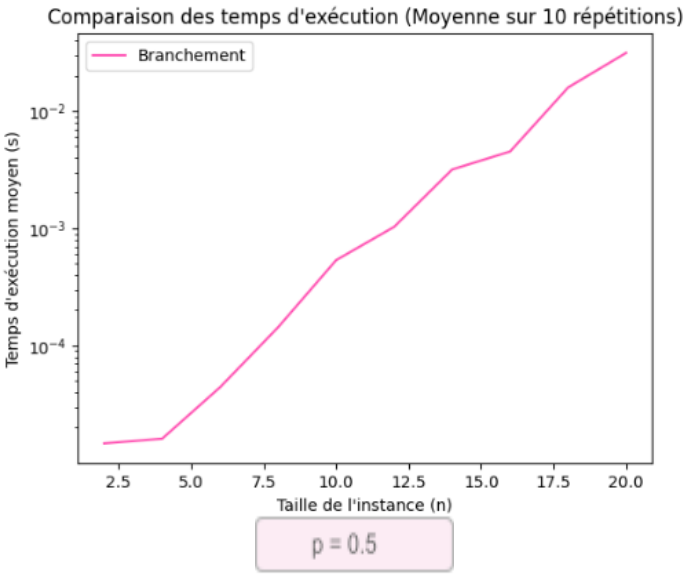
```
Meilleure solution: {'C', 'B', 'F'}
```

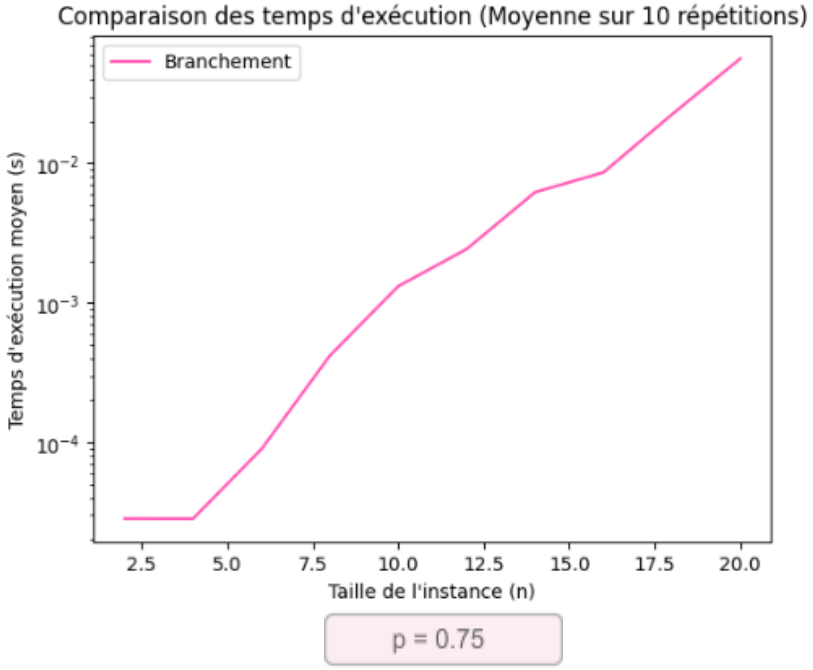



Graphe G1

Évaluons maintenant ses temps de calculs en fonction du nombre n de sommets et de la probabilité p.




On peut remarquer en comparant ces 3 graphiques que malgré la forme relativement similaire, plus la probabilité p augmente, donc plus il y a de chance d'avoir des arêtes, plus le temps d'exécution de branchement augmente également. Aussi, on constate que la fonction de branchement est nettement plus lente que les fonctions couplage et glouton. On atteint bien plus rapidement un certain seuil de temps, avec moins de sommets. Ici, et à vue d'oeil, on arrive à environ $10^{-2}$secondes à environ 17 sommets, temps qui n'est soit jamais atteint ( par couplage notamment) soit atteint pour bien plus de sommets ( environ 200 pour glouton).

## Ajout de bornes

Soit |C| ≥ max{b1, b2, b3} avec C une couverture d'un graphe G. Montrons que les bornes données b1, b2, b3 sont valides.

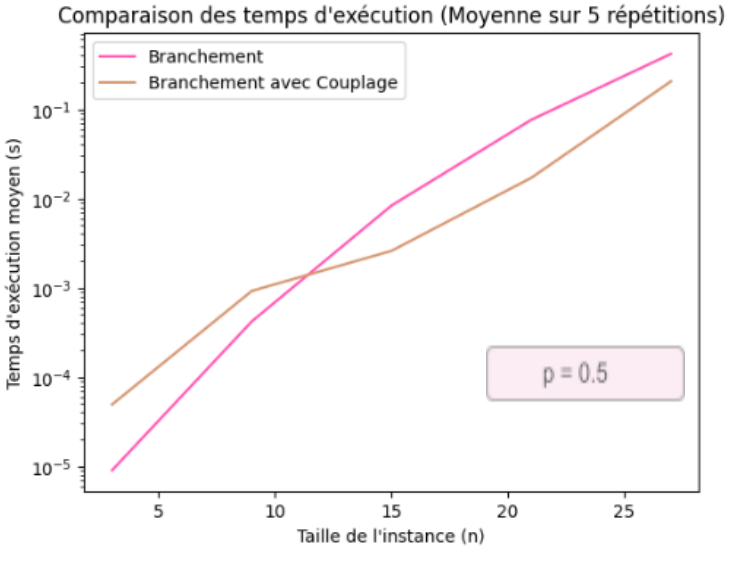
- Borne b1 = $\frac{m}{\Delta}$, où Δ est le degré maximum des sommets du graphe. Chaque sommet du graphe G a au plus Δ voisins, car Δ est le degré maximum. On a donc |C| x Δ $\geq m$ qui implique donc |C| $\geq \frac{m}{\Delta}$ donc b1 est valide.
- Borne b2 = |M|, où |M| est la taille d'un couplage du graphe. Cette borne repose sur la taille et la définition même d'un couplage de graphe. Un couplage est un ensemble d'arêtes non adjacentes, c'est-à-dire que les arêtes du couplage ne partagent aucun sommet en commun. Cette borne est valide car tout sommet qui appartient à un couplage doit être inclus dans la couverture minimale. Ainsi, la taille de la couverture minimale doit être au moins égale à la taille du couplage. On a bien |C| $\geq |M|$
- Borne b3 = $\frac{2n-1-\sqrt{(2n-1)^2-8m}}{2}$. Cette borne est un peu plus complexe. Elle est basée sur le nombre de sommets n et le nombre d'arêtes m du graphe. Posons |C| = x. On trouve alors 2 types d'arêtes différentes, celles dont les deux extrémités sont dans C et celles dont une seule des extrémités est dans C. Dans le premier cas, il y en a au plus $\frac{x(x-1)}{2}$ car c'est le nombre d'arêtes d'un graphe complet. Dans le second cas, il y en a au plus x(n-x) car chaque sommet de C est relié aux n-x autres sommets qui n'appartiennent pas à C. On peut ainsi avoir l'égalité suivante:

  $m \leq \frac{x(x-1)}{2} + x(n-x) \Leftrightarrow m \leq \frac{x^2-x+2nx-2x^2}{2} \Leftrightarrow m \leq \frac{-x^2+2(n-1)x}{2}$

  $\Leftrightarrow 0 \leq \frac{-x^2+2(n-1)x-2m}{2} \Leftrightarrow 0 \leq -x^2+2(n-1)x-2m$ En trouvant la racine de ce polynôme on a bien $\frac{2n-1-\sqrt{(2n-1)^2-8m}}{2}$= b3 et donc |C| $\geq b3$

En conclusion, chacune des bornes b1, b2 et b3 est valide et repose sur des propriétés spécifiques du graphe, comme le degré maximum, la taille du couplage et le nombre de sommets et d'arêtes.

Nous avons ensuite implémenté un algorithme de branchement partant d'une solution réalisable obtenue avec l'algorithme de couplage et calculant une borne inférieure à partir des 3 bornes citées plus haut. Voici une comparaison des temps d'exécutions de Branchement et de Branchement avec couplage.

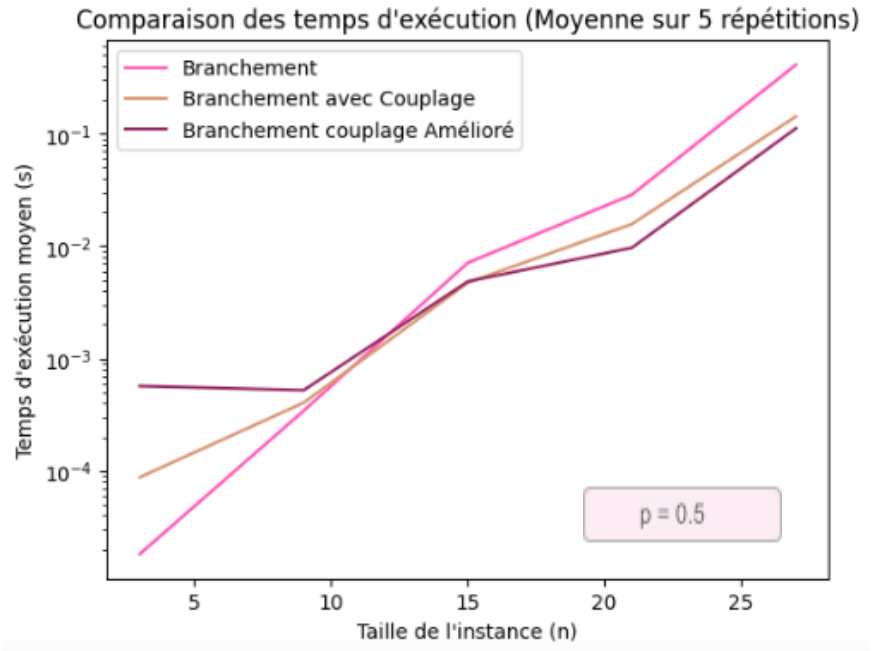



On constate que plus il y a de sommets, plus utiliser la méthode de branchement avec couplage est avantageux. En effet, pour un n assez petit, branchement simple est plus rapide car cela demande moins de temps de parcourir le graphe que d'appeler couplage. Cependant, une

fois un certain n atteint, ici environ 10, le temps d'exécution de branchement avec couplage est moins important que le branchement simple.

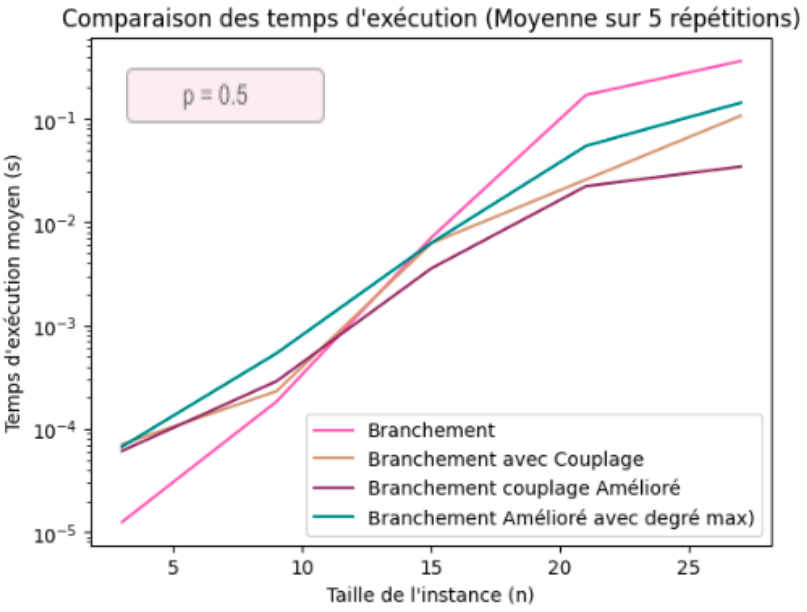
## Amélioration du branchement

Nous tentons, une fois de plus, d'améliorer l'algorithme de branchement, en reprenant le code de branchement avec couplage mais en modifiant légèrement sa façon de parcourir le graphe.




Pour les mêmes paramètres, on voit facilement que branchement amélioré est plus rapide que branchement avec couplage, une fois un certain n atteint. De même que pour les deux premières fonctions de branchement, il est préférable d'utiliser cette fonction améliorée lorsque le n est grand ( ici supérieur à 10 environ).

Une autre méthode à essayer afin de diminuer le temps de calcul lors de l'exécution de la fonction est de choisir le sommet en fonction de son degré dans le graphe. Nous avons implémenté cette fonction et voici ses temps d'exécution en comparaison avec les 3 premiers algorithmes de branchement.



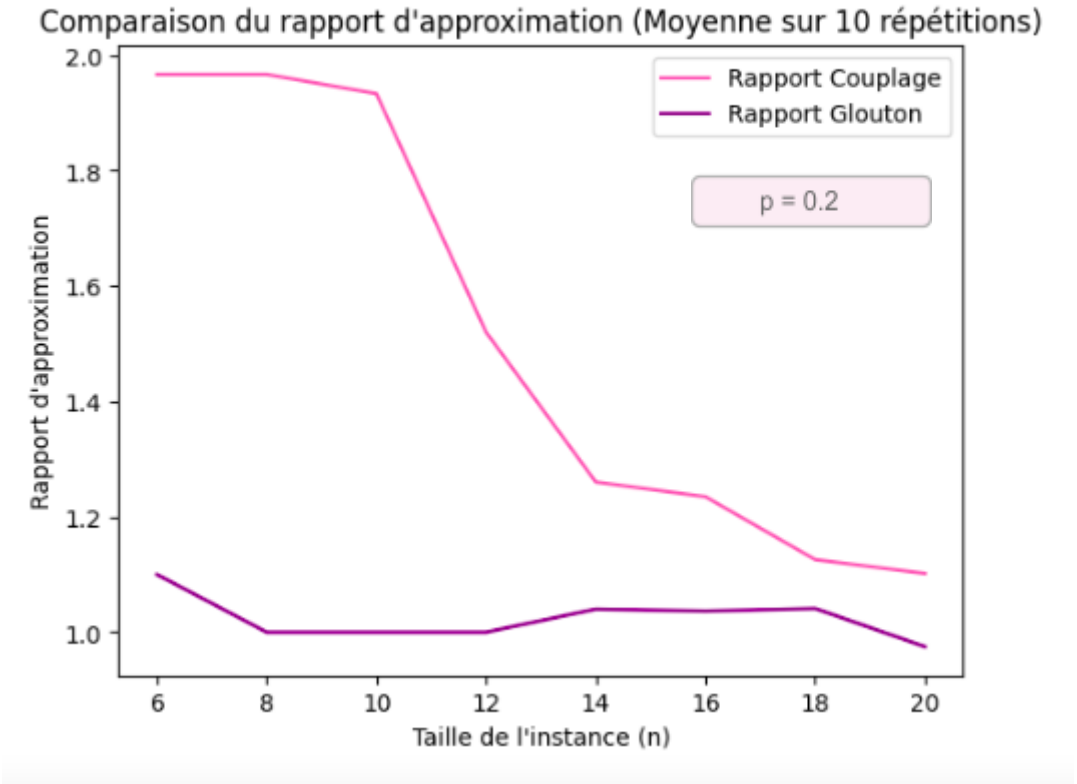

D'après nos expérimentations, branchement amélioré avec l'utilisation du degré maximum est plus lent que les 2 autres algorithmes de branchements améliorés. Il reste cependant plus rapide que l'algorithme de branchement de base à partir d'un certain n. L'incohérence sur les temps d'exécutions peut être due à une erreur d'implémentation de notre part. On pourrait également se poser la question au vu de la forme de la courbe, est ce que l'algorithme utilisant le degré maximum n'est pas plus rapide que les deux autres une lorsqu'il y a bien plus de n. Nous n'avons pas pû tester au delà de 30 car le temps d'exécution était bien trop élevé.

## Qualité des algorithmes approchés

Afin de comparer glouton et couplage nous avons cherché à comparer leurs rapports d'approximation, à savoir, en fonction de n, la taille de la solution de l'algorithme divisée par la taille de la solution optimale. Également, il nous retourne le pire rapport d'approximation pour chacun des deux algorithmes. Afin d'avoir une réponse claire, nous avons décidé d'ommettre les rapport retournant "inf" notamment lorsque la solution optimale était égale à 0.

Ici, nous constatons très clairement que l'algorithme de couplage a un rapport d'approximation bien plus élevé que l'algorithme glouton. On s'aperçoit aussi que le pire rapport de Couplage est 1.966 , ce qui confirme donc l'énoncé qui nous dit qu'il est 2-rapproché. Puis, le pire rapport de glouton est 1.1 dans cet exemple, ce qui nous rapproche aussi de notre résultat trouvé dans la partie 3 qui était que l'algorithme glouton est 1.2-rapproché.

```
Pire rapport d'approximation pour le couplage : 1.9666666666666668
Pire rapport d'approximation pour le glouton : 1.1
```




# Conclusion

En conclusion, de nombreuses méthodes permettent de trouver une couverture d'un graphe. On trouve des algorithmes approchés et des algorithmes exactes comme les algorithmes de branchement qui peuvent être améliorés en ajoutant des bornes ou encore une solution réalisable à partir d'un algorithme de couplage.