Rapport projet UE Complex 2021-2022

T.Bersoux

Version : 24 octobre 2021

Table des matières

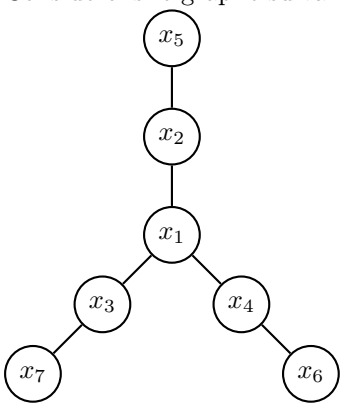
1 Graphes, opérations de bases

La structure proposée est un dictionnaire, dont les clés sont les sommets, et dont les valeurs sont les adjacences. Une fonction edges(G) a été ajoutée afin d'obtenir une liste d'arêtes, plus simple pour itérer par la suite. Elle est linéaire en nombre d'arêtes, et les tests ont montrés que son temps d'exécution était négligeable par rapport au temps total des fonctions qui l'utilisent.

2 Méthodes approchées

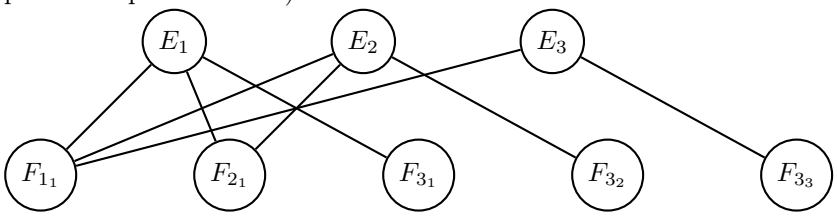
Glouton pas optimal, pas r-approché quelque soit r

Considérons le graphe suivant. Une solution optimale est $C = \{x_2, x_3, x_4\}$, de taille 3 :



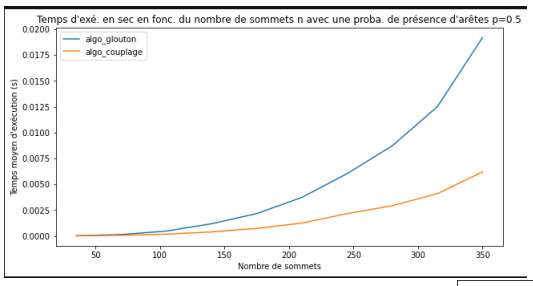
Pourtant, si l'on applique l'algorithme glouton sur ce graphe, il va prendre en priorité x_1 d'arité 3. Ensuite, il devra prendre $\{x_5, x_6, x_7\}$ afin de compléter la couverture, ce qui fera un ensemble de taille 4. Ainsi, la solution que le glouton retourne n'est pas optimale, donc le glouton n'est pas optimal. □

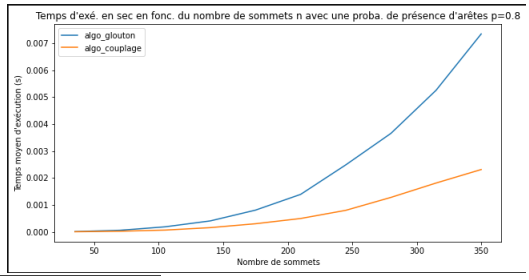
Considérons maintenant un graphe biparti $G = (V, E)$, c'est à dire tel que V se divise en deux ensemble E et F, et prenons le tel qu'il y ait n sommets dans E, et que F se divise lui même en n ensembles $F_1, F_2..F_n$, que chaque sommet de F_i ait exactement i voisins dans E, et que deux sommets de F_i n'aient pas de voisin en commun dans E.Avec cette construction, il y aura $\lfloor \frac{n}{i} \rfloor$ sommets dans tout F_i.
Subséquemment, chaque sommet de F_i est de degré i, et chaque sommet dans E sera de degré au plus n. (Tout petit exemple ci dessous)

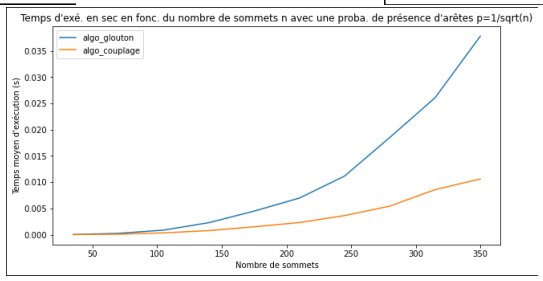


L'algorithme glouton peut donc d'abord prendre une arête dans F_n, puis dans F_{n-1}... jusqu'à ce qu'au final, il renvoie F comme couverture, alors que E est une couverture bien plus simple et optimale.
Le nombre de sommet dans la couverture donnée par le glouton est alors $\frac{|F|}{|E|}$, c'est à dire environ $log(n)$, qui tend vers l'infini quand n tend vers l'infini. Ainsi, glouton n'est pas r-approché quel que soit r. □

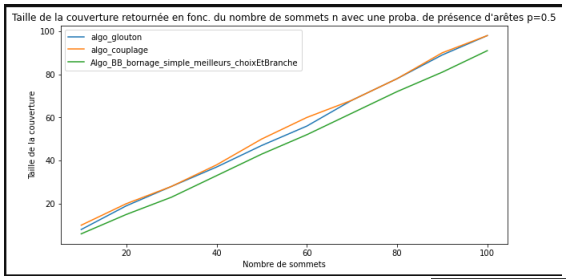
2.1 Tests numériques

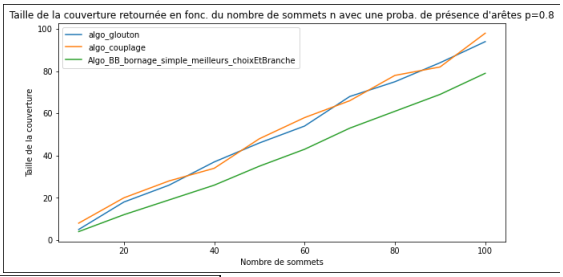


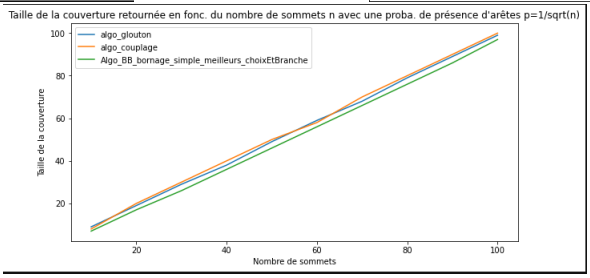




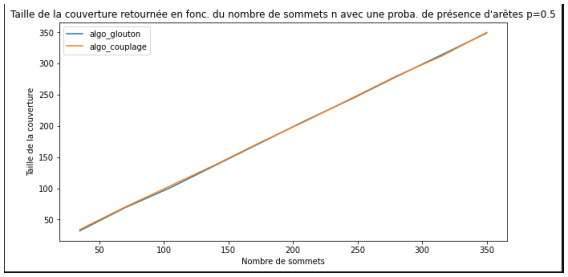
Ces deux algorithmes approchés s'exécutent très rapidement, même sur de grands graphes. En comparant les figures, on remarque que les graphes les plus denses(p=0.8) sont plus simples à traiter pour ces deux algos que les graphes moins denses(p=0.5,p=$\frac{1}{\sqrt{n}}$).







Bien qu'inexacts (ces algorithmes approchées sont ici comparés qualitativement avec un algorithme de branch&bound retournant la solution optimale), ils sont relativement proches de la solution optimale, et sont bien plus rapides quand n s'accroît. Comme vu sur la figure 7, il n'y a cependant pas vraiment de différence de qualité entre algo_couplage et algo_glouton. Expérimentalement, on peut considérer que ces algorithmes sont donc des approximations robustes, et peuvent être utilisés pour des calculs de bornes pour des approches exactes, ou pour des estimations de la solution optimale. Il arrive souvent, comme dans l'image suivante, de retrouver quasi parfaitement les même taille de couverture pour les deux algorithmes :



3 Séparation et évaluation

3.1 Branchement

J'utilise ici une version itérative, et non pas récursive, pour plusieurs raisons :

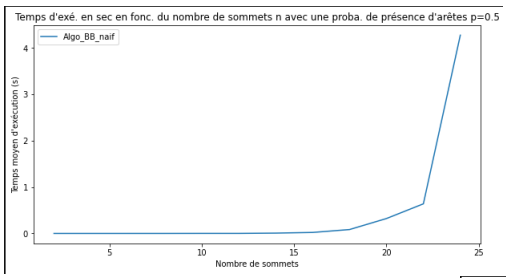
— On évite de nombreuses copies profondes et on se simplifie les retours : l'algorithme est donc plus efficace et plus clair.

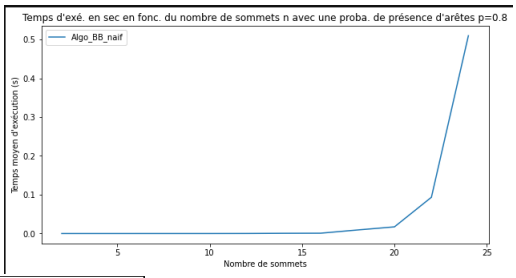
— La consigne parle d'utiliser une pile. On pourrait l'interpréter comme "pile récursive", mais je comprends plutôt "gérer la pile *à la main*".

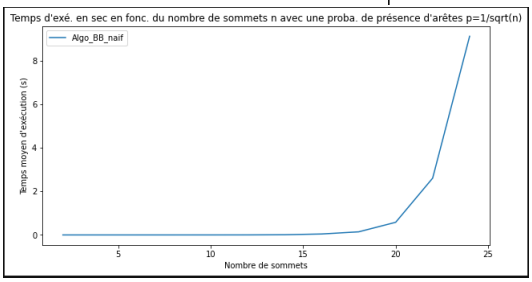
— Je n'ai jamais fait de DFS itératif, alors que j'ai déjà fait plusieurs DFS récursifs. Autant apprendre une nouvelle manière !

3.1.1 Tests numériques

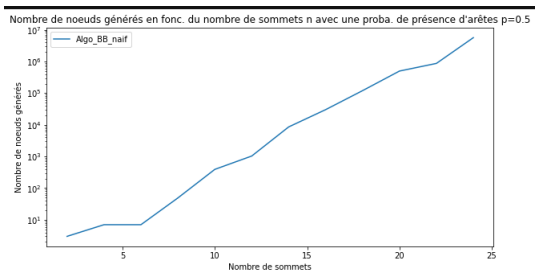
Cet algo est très *très* lent, car le nombre de solutions explorées est exponentiel :

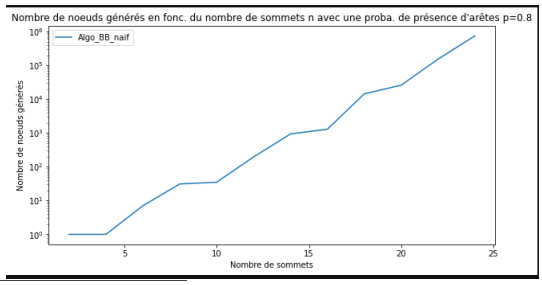


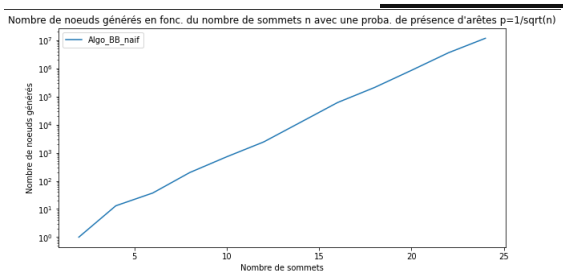




Pour un graphe de 20 sommets, on arrive à 10^6 noeuds ! Là aussi, les cas plus difficiles semblent être les graphes peu denses.







3.2 Ajout de bornes

Je n'ai pas tenu compte de la borne b_2, car je ne comprends pas s'il faut que M soit un couplage maximal ou non. De plus, s'il en est un, alors sa taille sera aussi la taille de la couverture minimal ? Et s'il n'en est pas un, comment le choisir ? Faut-il prendre le minimum et ainsi faut-il résoudre un autre problème qui est de trouver ce couplage minimal ? Dans tous les cas, je me suis abstenu de prendre en compte cette borne plutôt que de faire des erreurs et détruire mes fonctions de branch&bound(Au départ j'utilisais algo couplage, avant de me rendre compte que dans ce cas ma borne inf et ma borne sup étaient toujours égales. Je m'en suis rendu compte le jour du rendu, trop tard pour poser toutes les questions précédentes !).

Validité des bornes

- Supposons $|C| < \lceil \frac{m}{\Delta} \rceil$, alors $|C| < \frac{m}{\Delta}$ et donc $|C| \times \Delta < m$. Soit $i \in [\![1, |C|]\!]$.
  Soit $S_i \in C$,α_i le nombre d'arêtes "couvertes" par le sommet S_i. Alors :
  - Si $deg(S_i) = \Delta, \alpha_i = \Delta$
  - Si $deg(S_i) < \Delta, \alpha_i < \Delta$
  - (Si $deg(S_i) > \Delta$ est impossible)

  On a donc $\sum \alpha_i \leq \Delta \times |C|$. Or, par hypothèse, on a $\Delta \times |C| \leq m$, Mais comme C est une couverture (= toute arête est couverte), on a $\sum \alpha_i = m$, donc $\sum \alpha_i \leq \Delta |C| < m$, ce qui est une contradiction.
  Ainsi, on a nécessairement $|C| \geq \lceil \frac{m}{\Delta} \rceil = b_1$
- Une couverture C de G doit couvrir toutes les arêtes de M, et donc contenir un sommet de chacune de ces arêtes. Puisque M est un couplage, aucune de ses arêtes ne partage de sommet. Ainsi, on sait que C devra contenir au moins un sommet par arête de M, c'est à dire |M| sommets. On a alors, $|C| \geq |M| = b_2$

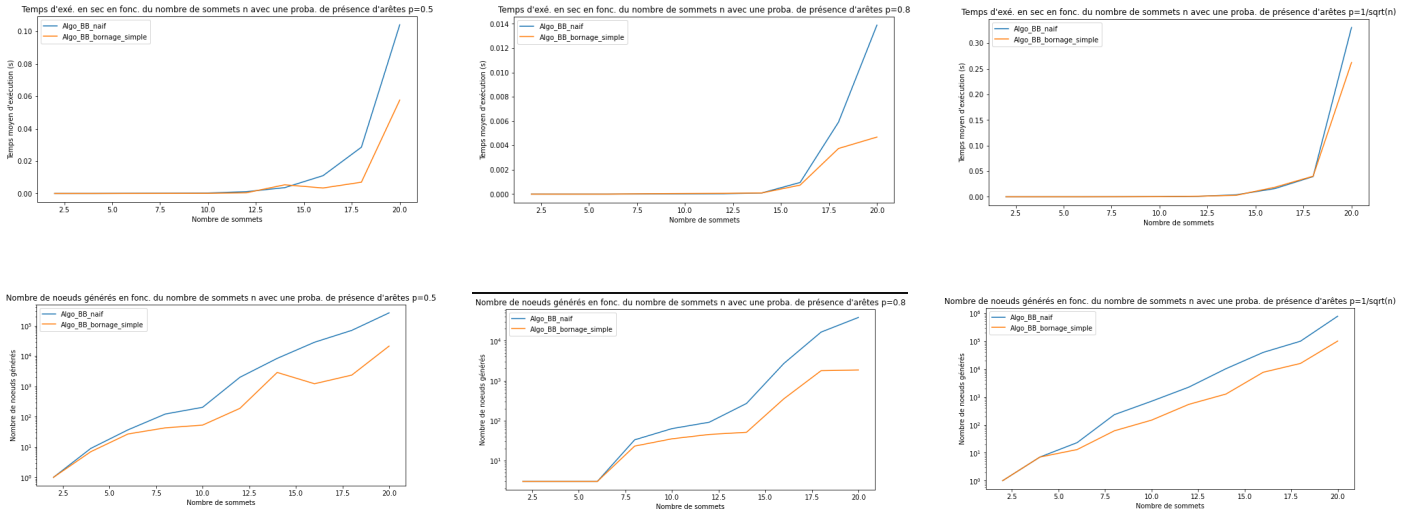
— Le nombre d'arêtes dans C est maximal quand, pour chaque arête, seulement un sommet est contenu dans C, et quand G est un graphe complet. On peut dénombrer ces arêtes dans ce cas : Le premier sommet S_1 de C est lié à |C|-1 sommets, donc il y a |C|-1 arêtes pour ce sommet. Le deuxième sommet S_2 sera aussi lié à |C|-1 sommets, mais il faut retrancher à ce compte l'arête (S_1, S_2), et il y aura donc |C|-2 nouvelles arêtes, et ainsi de suite. On peut alors compter le nombre total d'arêtes maximum dans C (Qui sera, de plus, supérieur ou égal à m puisque on est dans le cas où G est complet et donc avec le nombre maximum d'arêtes) :

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

$\Leftrightarrow 2m \leq |C| - (2n-1)|C| \Leftrightarrow |C| - (2n-1)|C| + 2m \leq 0$
On reconnaît alors un polynôme du second degré. Ses racines sont :
$x_1 = \frac{2n-1-\sqrt{(2n-1)^2-8m}}{2}$ et $x_2 = \frac{2n-1+\sqrt{(2n-1)^2-8m}}{2}$
Le discriminant et le terme de degré deux sont positifs donc si on veut que l'inégalité soit vérifiée, il faut que |C| soit entre les racines. On a donc :
$x1 \geq |C| \geq x_1 = \frac{2n-1-\sqrt{(2n-1)^2-8m}}{2} = b_3$

On a donc $|C| \geq b_1, |C| \geq b_2, |C| \geq b_3$, et donc $|C| \geq max(b_1, b_2, b_3))$ □

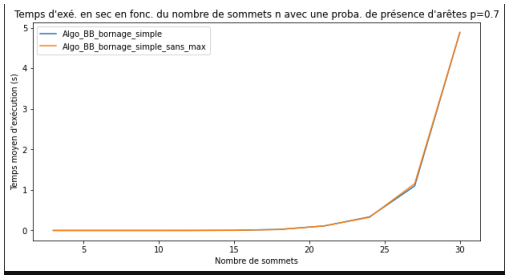
3.2.1 Tests numériques

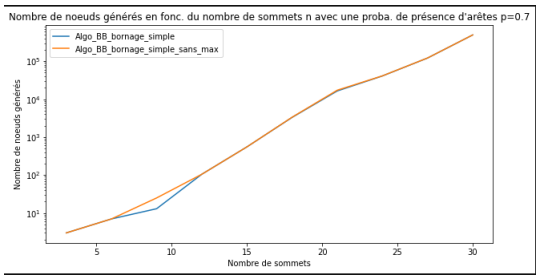


L'ajout de bornes (et donc d'élagage dans l'arbre des solutions) a grandement augmenté l'efficacité de l'algorithme. Il explore parfois 10 fois moins de nœuds, et gagne donc beaucoup en vitesse. La remarque sur le lien entre difficulté et faible densité reste valide.

Utiliser le Glouton ? Puisque le glouton est plus lent que l'algorithme donnant le couplage max, et que la qualité de ses solutions est la même, il n'y a pas d'intérêt à l'utiliser.

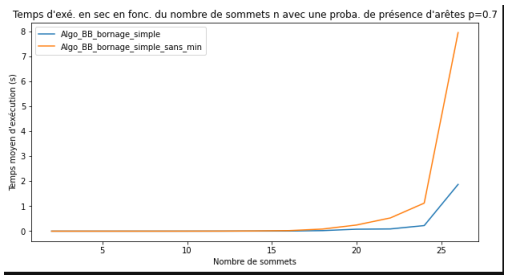
Utiliser seulement les bornes inférieures ?

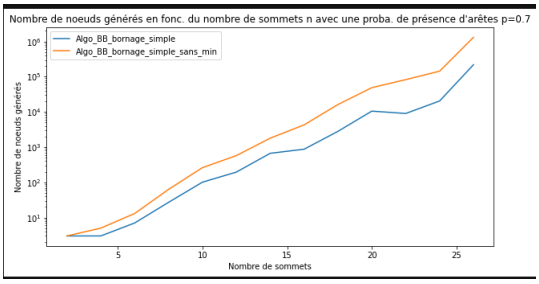




Comme on peut le constater avec ces tests, il n'y a pas de différence majeure, que ce soit dans le temps d'exécution ou dans le nombre de noeud. Cela s'explique par le fait que l'on utilise la borne max afin de savoir si, dans la branche en court, on est *forcément* plus optimal. Au mieux, on pourra alors potentiellement éliminer des sous branches de cette branche ce qui, au vu de la complexité de cet algorithme, n'aura pas grande influence.

Utiliser seulement le calcul d'une solution réalisable ?

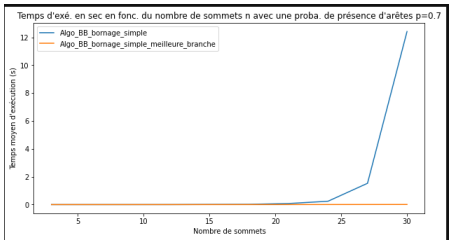


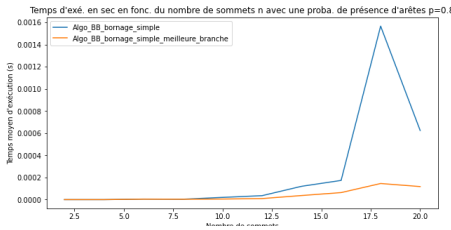


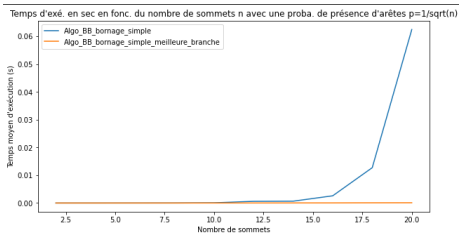
Dans ce cas par contre, l'efficacité est grandement diminuée : C'est normal, on élague beaucoup moins de branches lors de l'exploration !

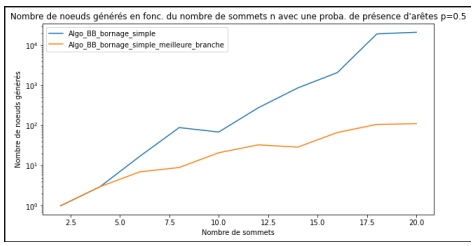
4 Amélioration du branchement

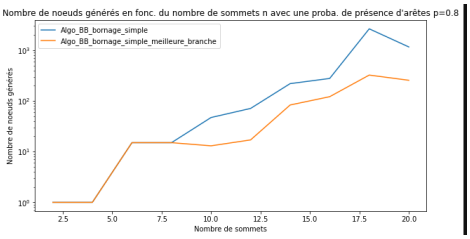
4.1 Amélioration 1

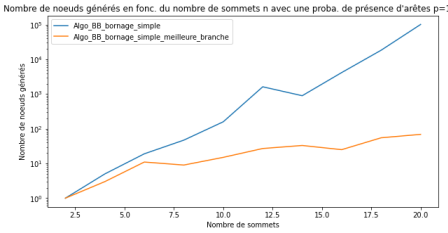




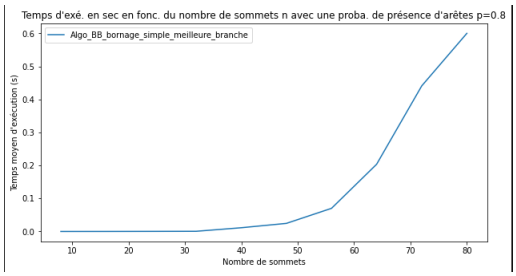


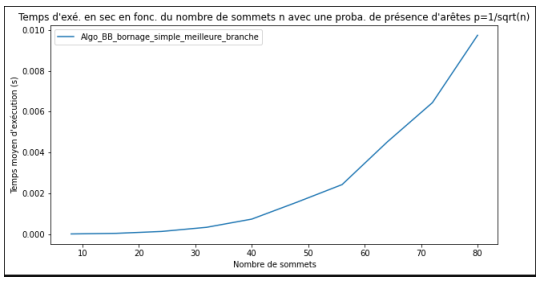




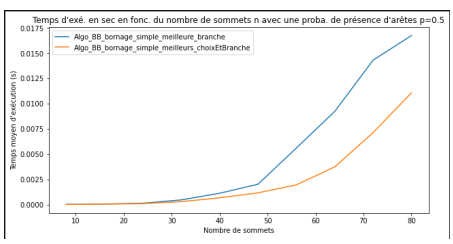


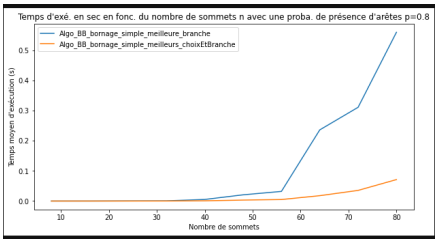
Avec cette amélioration, il y a encore un gain conséquent ! On diminue d'un facteur 1000 le nombre de nœud explorés. Cela nous permet maintenant de résoudre assez rapidement des instances d'une centaine de sommets environ. De plus, avoir un graphe dense nous arrange, puisque l'on priorise les sommets de haut degrés pour supprimer de nombreuses solutions sous-optimales. Ainsi, on est maintenant plus rapide sur les graphes denses que sur les graphes peu denses ! Ci dessous, un exemple avec des instance allant jusqu'à 80 sommets :

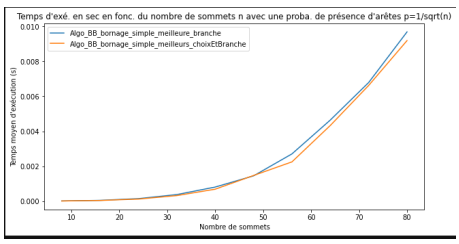


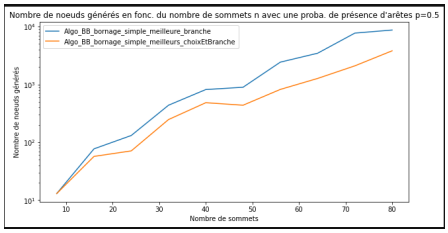


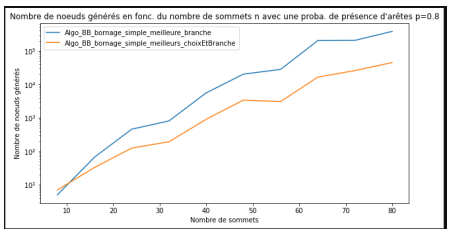
4.2 Amélioration 2

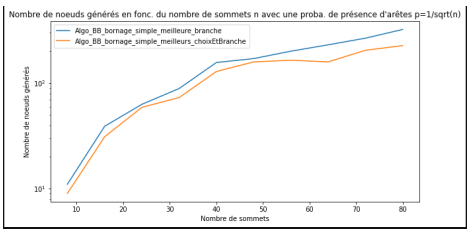




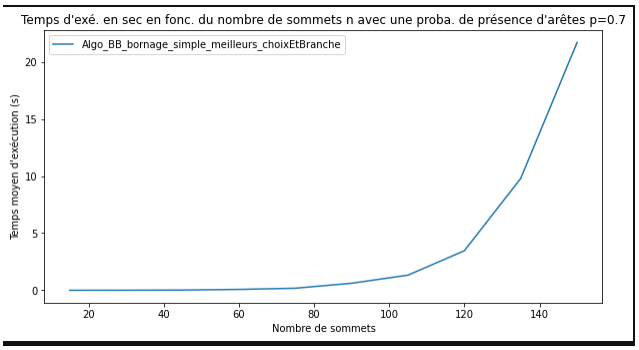








On rajoute encore une priorité sur les degrés, et cela se voit dans la différence de rapidité entre l'algorithme précédent et celui-ci. En effet, c'est pour des graphes denses que le gain est le plus élevé. Sur ce type de graphe, on arrive à traiter les instances de taille 100 en un peu plus d'une seconde, et on peut facilement monter sur des instances de taille 120 en une poignée de secondes, comme on peut le constater ci dessous :

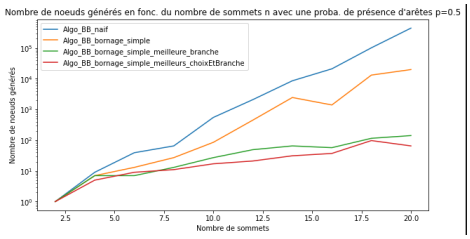


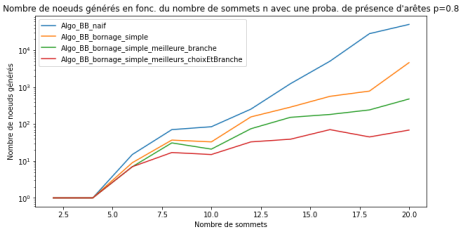
4.3 Amélioration 3

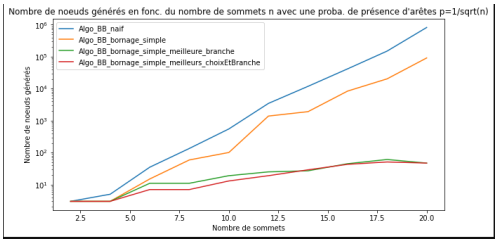
Pas eu le temps de l'expérimenter, mais il existe en effet toujours une solution sans u. En effet, pour la seule (car deg(u)=1) arête (u,v), on a deux possibilités : Soit v est de degré supérieur, soit v est de même degré. Dans les deux cas, une couverture avec v et non u est optimale.

4.4 Taille des arbres de décision

Ces figures mettent en valeur les ordres de grandeur de différence entre les divers algorithmes de branch and bound, en fonction de la densité des graphes :







5 Qualité des algorithmes approchés

(Pas assez d'expérimentations sur cette partie, je pense que mes résultats sont très peu significatifs)
Comme indiqué précédemment, tous deux renvoient *presque* la même taille de couverture. Si on compare avec l'algorithme de branch and bound muni de toutes les améliorations proposées, on trouve expérimentalement (Voir partie 2) que ces algorithmes approchés le sont d'un rapport $r \in [0.7, 0.9]$, en fonction de la densité du graphe. Le pire rapport que j'ai pu constater lors des expérimentations était un rapport un peu plus grand que 2, lorsque le graphe n'a que très peu d'arêtes (une ou deux par sommet, pas plus).

6 Ce que j'aurai voulu faire de plus...

Si j'avais plus de temps à consacrer à ce projet, il y a diverses choses que j'aurai voulu faire :

— Comprendre comment calculer la borne b_2, et regarder si elle est utile.

— Implémenter l'amélioration 3 pour le branch and bound.

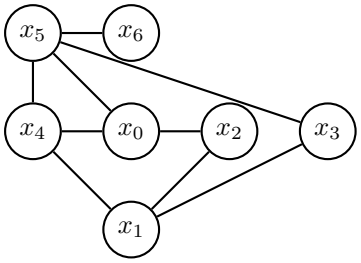
— Étudier plus en détais, sur de plus gros jeux de données, les résultats de Algo_couplage et du glouton : j'ai toujours du mal à croire que les deux renvoient presque les mêmes tailles de couverture.

— Faire des expérimentations numériques plus importantes, sur de plus gros jeux de données.

— Trouver d'autres heuristiques.

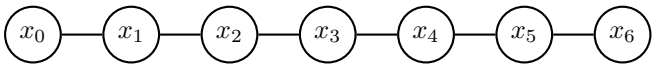
A Exemples de graphes simples utilisés

Exemple 1
Donné sur moodle



Une solution optimale est $C = \{x_0, x_1, x_5\}$, de taille 3

Exemple 2
Pour un graphe filiforme, algo_couplage devrait renvoyer une solution prenant tous les sommets.

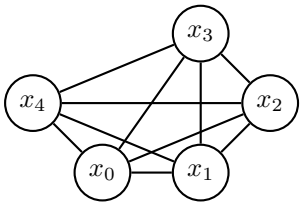


Une solution optimale est $C = \{x_1, x_3, x_5\}$, de taille 3
Sous format lisible par l'algorithme :

```
Nombre de sommets
7
Sommets
0
1
2
3
4
5
6
Nombre d aretes
7
Aretes
0 1
1 2
2 3
3 4
4 5
5 6
```

Exemple 3

Pour un graphe complet, tout algorithme devrait renvoyer une solution prenant tous les sommets.

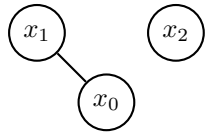


Une (la) solution optimale est $C = \{x_0, x_1, x_2, x_3, x_4\}$, de taille 3

Sous format lisible par l'algorithme :

```
Nombre de sommets
5
Sommets
0
1
2
3
4
Nombre d aretes
10
Aretes
0 1
0 2
0 3
0 4
1 2
1 3
1 4
2 3
2 4
3 4
```

Exemple 4
Quand un sommet est disjoint de tout autre, il ne devrait pas être pris dans les solutions de n'importe quel algorithme.



Une solution optimale est $C = \{x_0\}$, de taille 1
Sous format lisible par l'algorithme :

```
Nombre de sommets
3
Sommets
0
1
2
Nombre d aretes
1
Aretes
0 1
```