Projet M1 Androide
Bornes inférieures pour le Partitionnement des Graphes

Encadrants : Viet Hung Nguyen
Étudiants : Lucas Berterottière et Marc-Vincent Pereira

27 mai 2018

Table des matières

1 Introduction

1.1 Explication

Le projet sur lequel nous avons travaillé porte sur le k-partitionnement (avec k >2), c'est un problème qui consiste à diviser un graphe en k partie. Dans notre projet , nous ne travaillons que avec des graphes creux ($m = O(n)$ avec m le nombre d'arête des graphes),connexe et non-orienté.
les "clusters" doivent être crée de tel sorte que la somme des poids des arêtes dont les deux sommets appartiennent à un cluster différent soit minimum. On appelle ces arêtes les arêtes inter-cluster. Le but de notre projet est de trouvé des bornes inférieures qui soit à la fois accessible en un temps raisonnable et aussi le plus proche possible de la solution exacte.
Pour parvenir à notre but nous avons d'abord codé des routines nous permettant de générer des graphes creux, connexe et avec des poids qui suivent des distribution de probabilité particulière , ensuite nous avons essayé de trouver une solution exacte pour tous k-partitionnement de graphe et enfin nous avons étudié des algorithmes nous donnant une approximation de la solution avec des bornes inférieures selon la taille des graphes.

1.2 Application

Les problèmes de k-partitionnement de graphe et plus généralement de partitionnement de graphe trouvent leurs applications actuelles dans la planification des tâches dans des systèmes multi-processeurs, l'électronique, le calcul scientifique, mais aussi la modélisation des réseaux sociaux ou la bioinformatique.

2 Génération de graphe creux

2.1 Algorithme probabiliste

Data: $n \in \mathbb{N}$, $D \in [0,1]$
Result: $G = (V, E)$ où $|E| = O(|V|)$
$G = (\{1, ..., n\}, \emptyset)$ $L = [1, ..., n]$; `// la liste des noeuds non connectés au graphe`
$LP = [\frac{1}{n}, ..., \frac{1}{n}]$; `// liste de taille` n `qui associe à chaque noeud` i `la probabilité` $LP[i]$ `de connecter le noeud courant à ce noeud`
$d = 0$; `// la densité courante`
$c = 0$; `// le noeud courant`
while $|L|>0$ *ou* $d<D$ **do**
    ; `// i.e. tant que le graphe n'est pas connexe ou pas assez dense`
    on tire aléatoirement un indice i selon les probabilités de la liste LP;
    if $i \neq c$ **then**
        on ajoute l'arrête (c,i) au graphe;
        $c = i$;
        on enlève i de L;
        $norm = 1 - \frac{LP[i]}{2}$;
        LP[i]=LP[i]$\frac{}{2}$$\forall p \in LP : p \leftarrow \frac{p}{norm}$;
        $d \leftarrow d + \frac{1}{n^2}$;
Retourner G

Algorithm 1: Création d'un graphe creux par un algorithme probabiliste

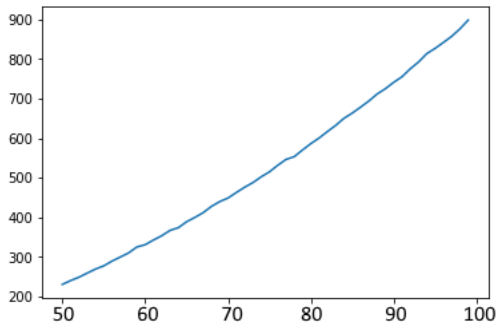


FIGURE 1 – Nombre d'arrêtes du graphe selon le nombre de sommets par l'algorithme 1

2.2 Preuve de la linéarité du nombre d'arrêtes

Terminaison La boucle Tant Que a deux conditions de sortie : il faut que le graphe soit connexe et que la densité soit supérieure ou égale à celle entrée en paramètre. Dans le pire des cas, si on ne regarde que ces conditions et sachant que l'algorithme rajoute ensuite des arêtes inexistantes, on peut être sur que le graphe sera connexe en n^2 ajouts d'arêtes et qu'à ce moment là, la densité étant de 1, elle est supérieure ou égale à celle rentrée en paramètre.

Complexité Nous allons étudier la probabilité qu'un sommet soit non connecté au graphe après un certain nombre d'itérations.
Tout d'abord nous allons considérer que les autres sommets ont une probabilité stable d'être sélectionnés, c'est à dire qu'à part le sommet qui nous intéresse, les autres sommets ne voient pas leur probabilité être mise à jour à chaque itération comme dans l'algorithme, mais qu'ils gardent leur probabilité initiale $\frac{1}{n}$. On peut se permettre cette aproximation car si le sommet qui nous intéresse a une probabilité P_s d'être sélectionné, qui augmente donc au fil des itérations où il n'est pas sélectionné, les autres sommets ont une probabilité $1 - \mathcal{P}$ à se partager.
Or comme $P_s > \frac{1}{n}$:
La probabilité moyenne pour un autre sommet d'être sélectionné sera :

$$\bar{p} = \frac{1 - P_s}{n - 1} < \frac{1 - \frac{1}{n}}{n - 1} = \frac{1}{n}$$

En augmentant la probabilité pour les autres sommets d'être sélectionnés on réduit celle de notre sommet, on augmente donc le nombre d'itération pour que $P_s = 1$. Nous allons donc fournir dans cette preuve une borne supérieur de la complexité moyenne de notre algorithme.
Dès lors, à une étape T de l'algorithme, la probabilité qu'un sommet qui ne vient pas d'être connecté au graphe est mis à jour en fonction de la probabilité p du sommet qui vient d'être relié au graphe comme suit :

$$P[T] \leftarrow \frac{P[T-1]}{1 - \frac{p}{2}}$$

En supposant que quelque soit le sommet sélectionné, $\bar{p} = \frac{1}{n}$, on aurait à un instant T la probabilité qu'un sommet soit connecté au graphe pour la première fois :

$$P[T] = \frac{P[0]}{(1 - \frac{\bar{p}}{2})^T} = \frac{\frac{1}{n}}{(1 - \frac{1}{2n})^T}$$

On cherche donc un nombre T d'itérations de la forme kn tel que $P[kn] \approx 1$. On pose :

$$(1 - \frac{1}{2n})^{kn} = e^{-\frac{k}{2}}$$

car c'est un équivalent en $+\infty$.
Dès lors :

$$P[kn] \approx 1 \Leftrightarrow \frac{\frac{1}{n}}{(1 - \frac{1}{2n})^{kn}} \approx 1$$

$$P[kn] \approx 1 \Leftrightarrow \frac{\frac{1}{n}}{e^{-\frac{k}{2}}} \approx 1$$

$$P[kn] \approx 1 \Leftrightarrow \frac{e^{\frac{k}{2}}}{n} \approx 1$$

$$P[kn] \approx 1 \Leftrightarrow e^{\frac{k}{2}} \approx n$$

$$P[kn] \approx 1 \Leftrightarrow k \approx 2\ln(n)$$

Ce qui nous donne donc un graphe ayant un nombre d'arêtes en $\mathcal{O}(n \log n)$, ce qui n'est pas ce qui était demandé puisque nous voulions un algorithme en $\mathcal{O}(n)$. Toutefois cette méthode garantit non seulement une bonne randomisation des graphes créés mais aussi un nombre de cycles intéressant par rapport à d'autres méthodes plus déterministes que l'on aurait pu utiliser. Cette dernière propriété va se montrer intéressante dans la suite du projet.

Nous avons cependant fait beaucoup d'arrondis supérieurs pour simplifier les calculs et la courbe de la figure 1 semble, quoi que légèrement convexe à vue d'oeil, presque linéaire. On considérera donc par la suite que cet algorithme nous fournit des graphes creux sachant que nous n'effectuerons nos test que sur des instances de taille faible (< 150 noeuds), nous pourrons donc toujours majorer le nombre d'arêtes du graphes par $5 * |V|$.

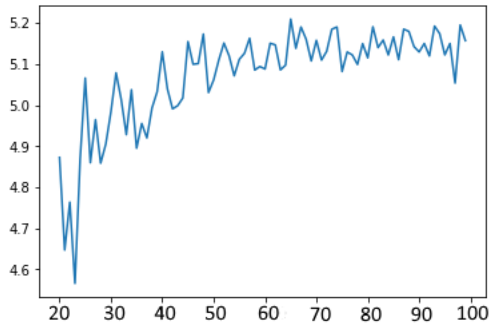


FIGURE 2 – Taille moyenne des cycles en fonction du nombre de sommets sur 10 graphes de densité 0.1

On constate que la taille moyenne des cycles semble se stabiliser juste au-dessus de 5.

2.3 Résultats

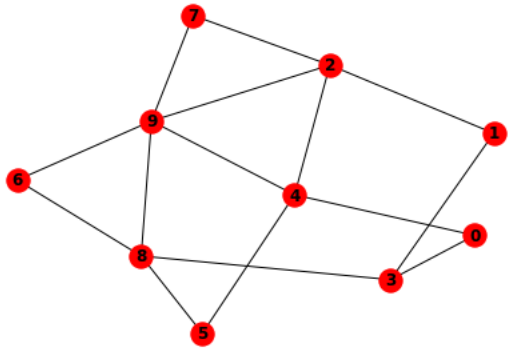


FIGURE 3 – Graphe creux généré par l'algorithme probabiliste avec 10 sommets et une densité de 0.2

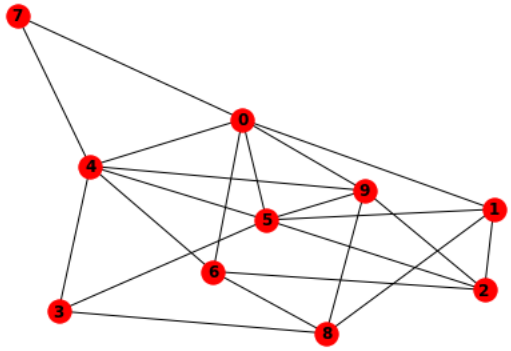


FIGURE 4 – Graphe creux généré par l'algorithme probabiliste avec 10 sommets et une densité de 0.3

La contrainte de connexité peut rendre la densité supérieur à celle entrée en paramètre, cependant, il semble selon la figure 1 que cela entre peut de fois en compte, notamment quand le graphe est suffisamment grand.

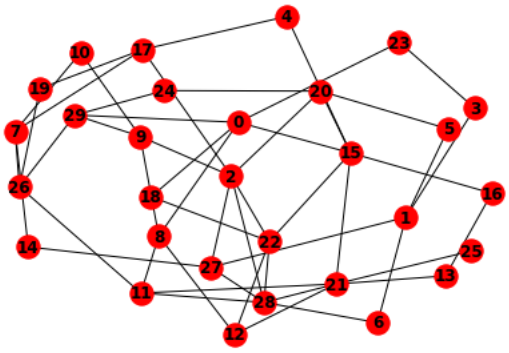


FIGURE 5 – Graphe creux généré par l'algorithme probabiliste avec 30 sommets et une densité de 0.05

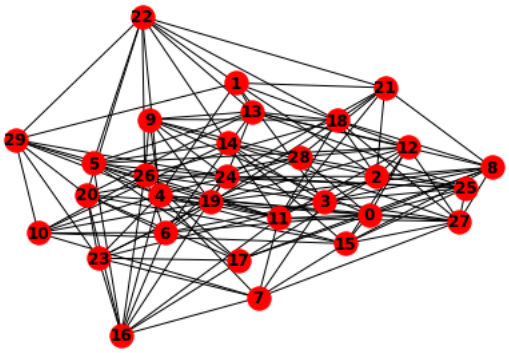


FIGURE 6 – Graphe creux généré par l'algorithme probabiliste avec 30 sommets et une densité de 0.2

2.4 Instanciation des poids sur les arrêtes

Optimisation en variable 0-1 Cette optimisation sera très utile pour tester le PLNE qui nous permettra de résoudre ce problème. Nous allons par la suite modéliser le problème de sorte à ce que une arrête (i, j) coupée par une frontière entre deux clusters soit égale à 1, 0 si au contraire les deux extremités de l'arrêtes appartiennent au même cluster.

Poids $\{-1, 1\}$ uniforméments distribués On instancie la première moitié des poids des arêtes à -1, la deuxième à 1. On utilise ensuite le mélange de Knuth pour obtenir une permutation aléatoire des poids des arêtes. Ce mélange garantit que chaque permutation est équiprobable.

Distribution gaussienne de poids entiers On initialise d'abord le poids de chaque arête par une variable aléatoire suivant une loi normale. On multiplie ensuite ces valeurs par un facteur 10^5 et on les convertit en entier. Comme au paragraphe précédent, on applique le mélange de Knuth pour obtenir une permutation aléatoire des poids des arêtes.

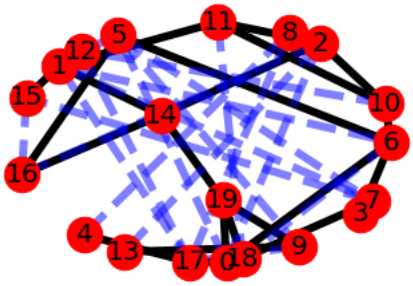


FIGURE 7 – Graphe creux avec des arrêtes de poids ± 1 (20 sommets, densité0.1)

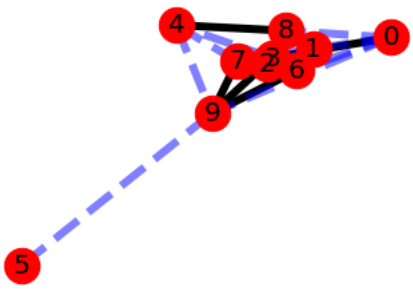


FIGURE 8 – Graphe creux généré grâce à la distribution gaussienne avec une loi normale de base $\mathcal{N}(10, 0.5)$ (10 sommets, densité=0.1, les arêtes noires appartiennent à $[10^6 - 10000, 10^6 - 10000]$

3 Résoudre le problème avec la programmation linéaire

Dans cette partie , nous donnons une solution exacte afin de résoudre le problème du k-partitionnement.

3.1 PLNE utilisé

Soit $G = (V, E)$ un graphe non-orienté connexe où $V = \{1, \ldots, n\}$ et $m = |E|$. À chaque arête ij, on associe une variable $x_{ij} \in \{0, 1\}$:

$$x_{ij} = \begin{cases} 1 & \text{si } ij \text{ appartient à l'inter-cluster,} \\ 0 & \text{sinon.} \end{cases}$$

à chaque sommet i, on associe une variable x_i :

$$x_i = \begin{cases} 1 & \text{si } i \text{ est le plus petit sommet (en terme d'indice) dans son cluster,} \\ 0 & \text{sinon.} \end{cases}$$

Soit $\mathcal{C}$ l'ensemble des cycle sans corde de G (chordless cycle en anglais). Soit $F \subset E$ et posons $x(F) = \sum_{e \in F} x_e$.
On veut diviser notre graphe en k-partitions de façon à minimiser la somme des arrêtes appartenant à l'inter-cluster ; la figure (9) est un exemple pour le 3-partitionnement.

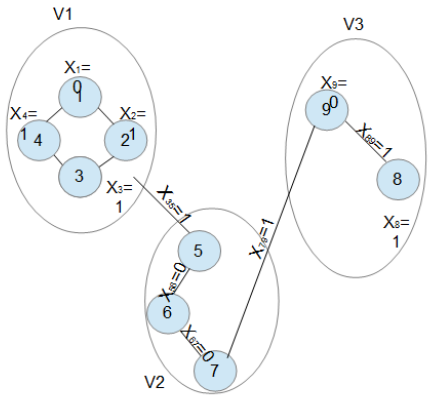


FIGURE 9 – 3-partition d'un graphe

Le programme linéaire 0/1 (PLNE) du k-partitionnement est le suivant.

$$\min \sum_{ij \in E} c_{ij} x_{ij}$$

$$x_e - x(C \setminus \{e\}) \leq 0 \text{ pour tout } C \in \mathcal{C} \text{ et pour tout } e \in C \quad (1)$$

$$x_j \leq x_{ij} \text{ pour tout } 1 \leq i < j \leq n \text{ et } ij \in E \quad (2)$$

$$x_j + d_{ij} - 1 + \sum_{1 \leq k < j et kj \notin E} x_k \geq \sum_{\substack{1 \leq i < j \\ ij \in E}} x_{ij} \text{ pour tout } j \in V \quad (3)$$

$$x_{ij} + x_{kj} \geqslant x_i + x_k - 1 \text{ pour tout } 0 \leqslant \text{i<k<j} \quad (4)$$

$$\sum_{i=1}^{n} x_i = k \quad (5)$$

$$x_e \in \{0, 1\} \text{ pour tout } e \in E$$

$$x_i \in \{0, 1\} \text{ pour tout } i \in V x_{ij} + x_{kj} \quad (6)$$

3.2 Contraintes

- Contraintes (1) sont les inégalités de cycle homogènes expliquée en 3.3.1 .
- Contraintes (2) disent que si $x_{ij} = 0$, i.e. j est dans le même cluster que i avec $i < j$ alors $x_j = 0$.
- Dans les contraintes (3), d_{ij} est une constante qui est égale au nombre d'arêtes $ij \in E$ t.q. $1 \leq i \leq j - 1$. Ces contraintes disent que si $\sum_{\substack{1 \leq i < j \\ ij \in E}} x_{ij} = d_{ij} + \sum_{1 \leq k < j et kj \notin E} x_k$ i.e. cette somme atteint son maximum et aucun sommet plus petit que j est dans le même cluster que j alors $x_j = 1$, i.e. j est le plus petit sommet dans son cluster.
- Contrainte (5) dit tout simplement que le nombre de clusters dans la partition est exactement k.

3.2.1 Contrainte de cycle

Les contraintes de cycle (contraintes (1)) sont appelée les inégalités de cycle homogène ; elles permettent de s'assurer que chaque sommet d'un cycle appartient soit à la même partition soit à à plusieurs mais dans ce cas il doit y avoir au moins deux coupes du cycle.
Par exemple si on prend un cycle coupé une seul fois comme sur la figure (10) , une contrainte de cycle homogène est alors violée puisque $x_{1,2}$-($x_{1,2}$+$x_{1,2}$+$x_{1,2}$)$\geqslant$0. Le cycle est donc coupé au minimum une deuxième fois comme sur la figure (11).

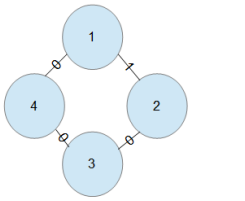


FIGURE 10 – cycle avec x_{ij} sur chaque arrête (i,j)

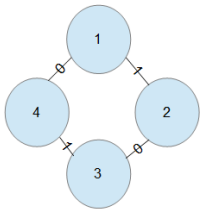


FIGURE 11 – cycle avec x_{ij} sur chaque arrête (i,j)

Ces contraintes sont obligatoire car si un cycle n'est coupé que une fois comme sur la figure (10), alors les sommets du graphe sont à la fois dans la même partition et dans des partitions différentes. Le programme linéaire est par contre difficile à résoudre à cause du nombre exponentiel de cycles sans cordes qui peuvent être présent dans les graphes, la figure (12) montre que plus le nombre de sommet dans un graphe est grand plus de nombre de cycle sans corde est important (en moyenne). On va donc essayer dans un premier temps de rajouter ces contraintes en "lazy constraints".

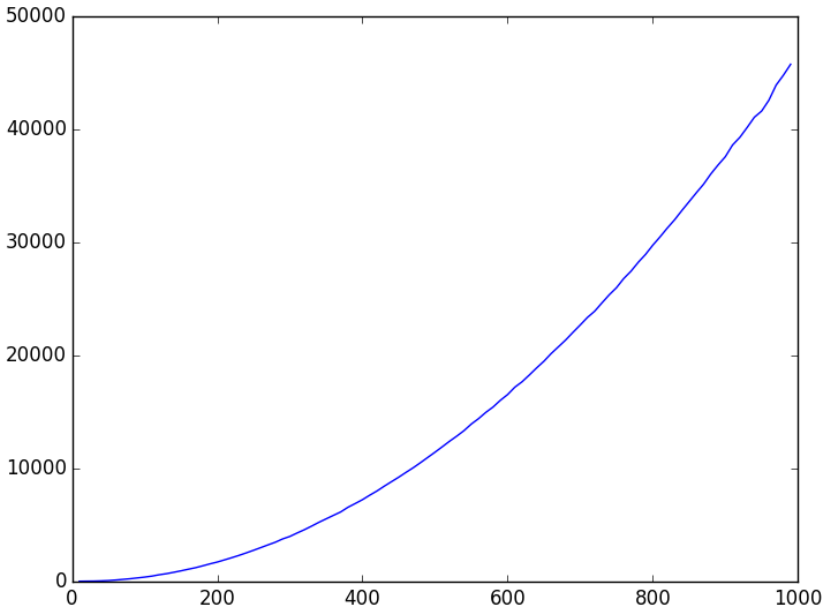


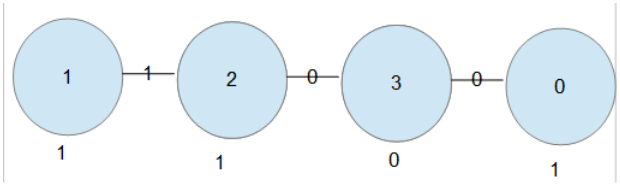
FIGURE 12 – nombre de cycle de la base en fonction du nombre de sommet dans un graphe creux

3.2.2 Problème

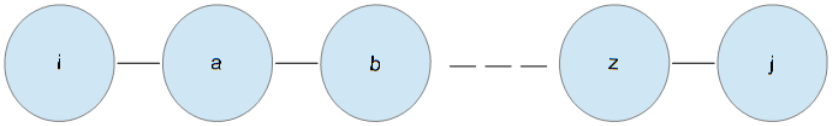
Lorsque l'on a codé le programme linéaire à l'aide de Gurobi ; on s'est aperçu que dans certain cas le résultat obtenu n'était pas celui espéré. Voici un des graphe pour lequel le programme linéaire codé ne nous donne pas un bon partitionnement :

Soit G=(V,E) ; tel que V={0,1,2,3} l'ensemble des sommets et E={(1,2) ;(2,3) ;(3,0)} l'ensemble des arrêtes du graphe G (toute les arrêtes ont un poids de 1).

Avec notre programme nous obtenons ce 3-partitionnement :

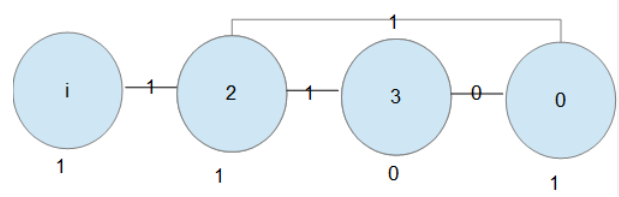


Les valeurs sur les arrêtes étant les x_{ij}, les valeurs à l'intérieur des sommets sont leur indice et à l'extérieur les x_i.
Les sommets 2 et 0 sont les sommets d'indice le plus petit de leur partition (cluster) puisque x_2=1 et x_0=1, pourtant ils appartiennent à la même partition car il existe un chemin Ch entre ces deux sommets tel que $\forall$ e $\in Ch$, x_e=0. On a donc une partition avec deux sommet d'indice minimale .
Plus généralement ce problème peut apparaître lorsqu'il existe , dans le graphe que l'on veut partitionner , un chemin entre deux sommet d'indice i et j de cette forme :



Avec i>a>b> .....>z et z<j.

Une solution pour résoudre ce problème serait de rajouter une arrête de poids 0 entre les sommets i et j ; dans ce cas la contrainte (2) du programme imposerait à x_{ij} d'être égal à 1 si x_i=1 et x_j=1 , de plus en rajoutant l'arrête entre i et j on crée un cercle et la contrainte (1) impose qu'il existe au moins une arrête e $\in Ch$, le chemin entre i et j, tel que x_e=1. La solution de programme linéaire reste inchangée puisque l'arrête rajoutée est de poids 0.



Le problème étant maintenant de reconnaître tout les chemins de ce type entre chaque sommet i,j du graphe. Dans le pire des cas ou si l'on décide de rajouter une arrête entre chaque sommet ; au total on aura rajouté $2 \times \binom{n}{2}$ avec n le nombre de sommet , ce qui donne n(n-1)contraintes

en plus au programme linéaire.On aura aussi un graphe complet , ce que l'on veut éviter puisqu'on souhaiterais rester avec des graphes creux.

Afin de résoudre ceci sans rajouter trop de contraintes à notre programme nous allons utiliser les "lazy constraints".

3.3 Lazy constraints

Les "lazy constraints" sont des constraintes qui ont peu de chance d'être violées lors de l'exécution de notre programme linéaire ; il est donc préférable de les rajouter seulement si elle sont importantes.
Le principe est simple ont va exécuter notre programme sans ces contraintes. On devrait alors obtenir une solution qui est une borne inférieure à la solution de notre problème ;ensuite on regarde si une de nos "lazy constraints" est violée ; si c'est le cas , on l'a rajoute au programme linéaire.
Pour voir si une de nos contrainte est violée ; on va d'abord faire le graphe G composé uniquement des arrêtes (a,b) tel que x_{ab}=0. On construit en fait le graphe non connexe de toute nos partition.
Si dans ce graphe G il existe un chemin entre deux sommet i et j et x_i=1,x_j=1 ; alors on a deux sommet d'indice minimum dans la même partition comme dans la partie 3.3.2, une contrainte est donc violée. on va alors rajouter la contrainte suivante :

Soit i,j deux sommet, soit Ch le chemin entre i et j dans le graphe G :

$$x_i + x_j - \sum_{e \in Ch} x_e \leq 1$$

Ainsi si x_i=1 et x_j=1 alors $\exists e \in Ch$, x_e=1. Cette contrainte permet de simuler en quelque sorte l'ajout d'une arrête de poids 0 dans le graphe pour résoudre le problème vu en 3.3.2 .

De même , on va regarder chaque arrête (i,j) tel que x_{ij}=1 ,Si il existe un chemin Ch entre i et j dans le graphe G, alors la contrainte de cycle homogène : $x_{ij} + \sum_{e \in Ch, e \neq (i,j)} x_e \leq 0$ est violée et on l'ajoute au programme.`

4 Étude bornes inférieures

Dans cette partie nous avons essayé de trouver des bornes inférieures pour des graphes trop grand pour être résolue par nos précédents algorithmes. Comme nous l'avons déjà vu sur la figure (12) , le nombre de cycle sans corde grandit exponentiellement par rapport au nombre de sommets , on va donc prendre des sous ensembles de ces cycles pour pouvoir limiter le nombre de contraintes.
Nous allons chercher une taille L au-delà de laquelle nous allons ignorer les cycles de la base. Cette relaxation va nous permettre de ne pas prendre en compte des contraintes coûteuses en temps pour un bénéfice négligeable en résultat.

4.1 Base de cycle

Un bon sous-ensemble de cycle peut être obtenu en prenant un sous ensemble de la base de cycle.Une base de cycle est un ensemble de cycle minimal a partir duquel on peut générer l'espace de cycle d'un graphe. Chaque cycle peut être représenté par leur vecteur caractéristique dans $\mathbb{R}^E$, avec E l'ensemble des arrêtes. par exemple le vecteur caractéristique du cycle [1,2,3] de la figure (13) est :
E=(1,2),(2,3),(3,4),(1,3),(4,1)
Vect([1,2,3])=[1,1,0,1,0] car le cycle [1,2,3] est composé des arrêtes (1,2),(2,3) et (1,3).

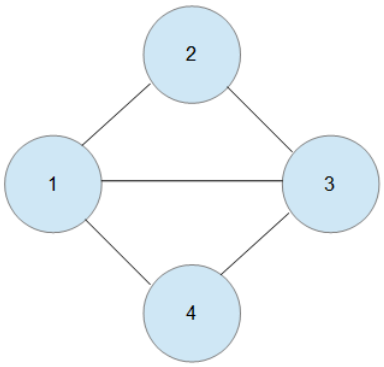


FIGURE 13

De même le vecteur caractéristique de chaque cycle d'un graphe peut être obtenu par combinaison linéaire des vecteur de cycle de la base modulo 2. Par exemple :
Vect([1,2,3])+Vect([1,4,3])=[1,1,0,1,0]+[0,0,1,1,1] = [1,1,1,0,1]mod 2
qui est le vecteur caractéristique du cycle [1,2,3,4]. A partir des cycle [1,2,3] et [1,4,3] on peut obtenir tout les cycle de la figure (13), donc ces deux cycles forme bien une base pur ce graphe.

Dans la suite de nôtre rapport , nous allons limiter les ensembles de cycle utilisé lors des contraintes grâce à la base de cycle d'un graphe et une constante L, la longueur maximale de chaque cycle dans l'ensemble.

4.2 Étude de comparaison de la qualité des bornes inférieures

Dans cette partie, les tests seront effectués avec des instances de graphe de n noeuds, de densité 0.1 et à diviser en $n/4$ clusters.

4.2.1 Étude des résultats en fonction de la taille maximale des cycles de la base

Nous avons observé l'évolution des résultats en fonctions de la taille L maximale des cycles de la base.

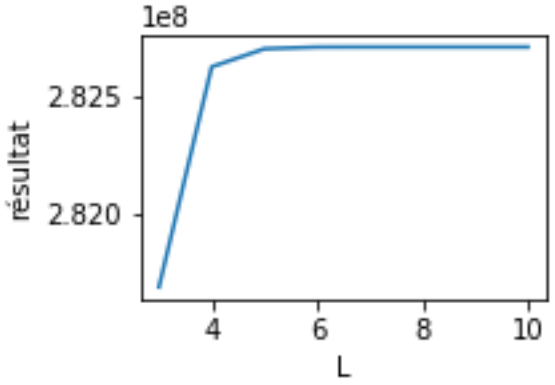


FIGURE 14 – Évolution des résultats retournés en fonction du L utilisé sur 100 graphes à 150 sommets de densité 0.1 (poids gaussiens)

On remarque que les résultats convergent rapidement dès $L = 5$, ce qui coïncide avec le résultat trouvé en figure 2 (partie 2.2).

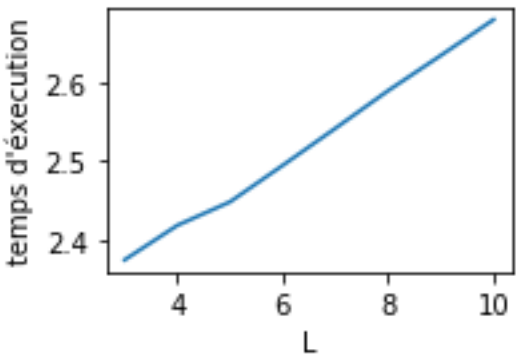


FIGURE 15 – Temps d'exécution du test ci-dessus en fonction de L

On remarque que le temps d'exécution semble être linéaire passé $L = 5$. Ces deux courbes montrent donc que passé $L = 5$, l'algorithme va gagner en temps de calcul pour un bénéfice négligeable.

4.2.2 Étude des résultats en fonction du nombre de noeuds

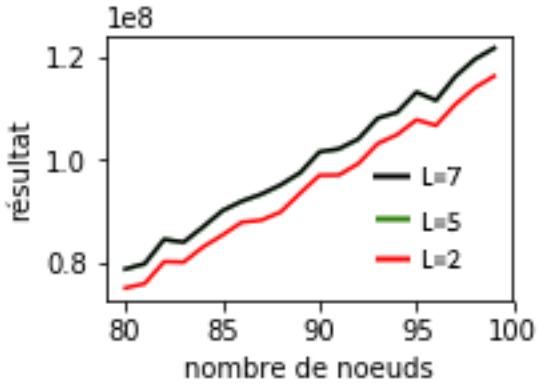


FIGURE 16 – Résultats en fonction du nombre de noeuds pour chaque fois 10 graphes de densité 0.1

On peut observer sur la figure 16 que les courbes pour $L = 5$ et $L = 7$ se superposent. La courbe pour $L = 2$ est quand à elle presque en tous points à une translation vectorielle des deux autres courbes.

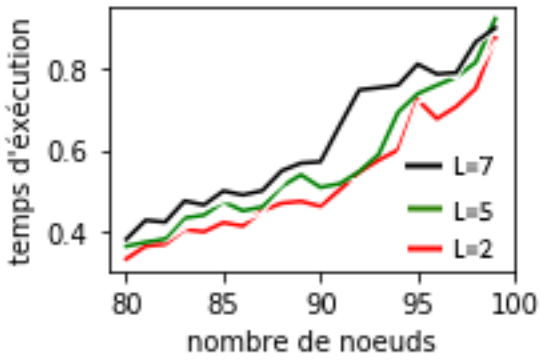


FIGURE 17 – Temps d'exécution en fonction du nombre de noeuds pour chaque fois 10 graphes de densité 0.1

Sur ces courbes on peut observer que les temps d'exécution ne sont pas linéaire par rapport à L comme on pouvait le supposer d'après la figure 15.

5 Conclusion

Le principal aspect de ce projet était d'implémenter et d'analyser les résultats d'une méthode de partitionnement de graphe en k clusters. Cette méthode repose sur les contraintes de cycle (3.2.1) qui garantissent l'exactitude de la solution. Toutefois, on a un nombre exponentiel de ces contraintes en fonction du nombre de sommets.

C'est pourquoi la méthode que nous avons utilisé ignore beaucoup de ces cycles, afin de pouvoir calculer une valeur approchée de la coupe minimale des k-clusters du graphe en temps polynomial.

Nous n'avons malheureusement pas pu établir de correspondance entre la valeur exacte de la coupe et la valeur approchée car le temps de calcul de la coupe par la méthode exacte était trop long sur des instances significatives.

Les résultats que nous avons obtenus dégagent une valeur intéressante de $L = 5$ qui est à la fois la valeur moyenne de la longueur des cycles des bases des graphes (Figure 2), la valeur de L à partir de laquelle les résultats de la méthode approchée n'augmentent plus significativement (Figure 14) et le seuil à partir duquel le temps d'éxécution est proportionnel à L (Figure 15).