

Mémoire

Pour Obtention du diplôme de Master En Informatique

Option : Système Informatique (SIQ)

Compression de Graphes par extraction de motifs et k2-trees : étude et implémentation

Réaliser par :

Mlle. Hafsa Bousbiat

eh_bousbiat@esi.dz

ESI

Mlle. Sana Ihadadene

es_ihadadene@esi.dz

ESI

Encadreurs :

Dr. Karima Amrouche

k_amrouche@esi.dz

ESI

Dr. Hamida Seba

hamida.seba@univ-lyon1.fr

Université de Lyon

Dr. Mohammed Haddad

mail

Université de Lyon

Octobre 2018

Année Universitaire : 2018-2019

Remerciement

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Nostrum tempore ea fugiat numquam autem saepe quas porro vitae? Fugit commodi tempore voluptate sint fugiat, possimus optio ad! Pariatur, obcaecati quidem. Lorem ipsum dolor, sit amet consectetur adipisicing elit. Neque excepturi ducimus accusantium eius voluptatibus, quod velit, explicabo tenetur aliquid ipsam sapiente. Quibusdam quis ullam, saepe numquam molestias nobis recusandae labore? Lorem ipsum dolor sit, amet consectetur adipisicing elit. Nostrum tempore ea fugiat numquam autem saepe quas porro vitae? Fugit commodi tempore voluptate sint fugiat, possimus optio ad! Pariatur, obcaecati quidem. Lorem ipsum dolor, sit amet consectetur adipisicing elit. Neque excepturi ducimus accusantium eius voluptatibus, quod velit, explicabo tenetur aliquid ipsam sapiente. Quibusdam quis ullam, saepe numquam molestias nobis recusandae labore? Lorem ipsum dolor sit, amet consectetur adipisicing elit. Nostrum tempore ea fugiat numquam autem saepe quas porro vitae? Fugit commodi tempore voluptate sint fugiat, possimus optio ad! Pariatur, obcaecati quidem. Lorem ipsum dolor, sit amet consectetur adipisicing elit. Neque excepturi ducimus accusantium eius voluptatibus, quod velit, explicabo tenetur aliquid ipsam sapiente. Quibusdam quis ullam, saepe numquam molestias nobis recusandae labore?

Résumé

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Nostrum tempore ea fugiat numquam autem saepe quas porro vitae? Fugit commodi tempore voluptate sint fugiat, possimus optio ad! Pariatur, obcaecati quidem. Lorem ipsum dolor, sit amet consectetur adipisicing elit. Neque excepturi ducimus accusantium eius voluptatibus, quod velit, explicabo tenetur aliquid ipsam sapiente. Quibusdam quis ullam, saepe numquam molestias nobis recusandae labore? Lorem ipsum dolor sit, amet consectetur adipisicing elit. Nostrum tempore ea fugiat numquam autem saepe quas porro vitae? Fugit commodi tempore voluptate sint fugiat, possimus optio ad! Pariatur, obcaecati quidem. Lorem ipsum dolor, sit amet consectetur adipisicing elit. Neque exceptu

Abstract

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Nostrum tempore ea fugiat numquam autem saepe quas porro vitae? Fugit commodi tempore voluptate sint fugiat, possimus optio ad! Pariatur, obcaecati quidem. Lorem ipsum dolor, sit amet consectetur adipisicing elit. Neque excepturi ducimus accusantium eius voluptatibus, quod velit, explicabo tenetur aliquid ipsam sapiente. Quibusdam quis ullam, saepe numquam molestias nobis recusandae labore? Lorem ipsum dolor sit, amet consectetur adipisicing elit. Nostrum tempore ea fugiat numquam autem saepe quas porro vitae? Fugit commodi tempore voluptate sint fugiat, possimus optio ad! Pariatur, obcaecati quidem. Lorem ipsum dolor, sit amet consectetur adipisicing elit. Neque exceptu

Table des matières

Remerciement	1
Résumé	2
Liste des figures	4
Liste des tableaux	5
1 Introduction	6
2 Théorie des graphes	7
2.1 Graphe non orienté	8
2.1.1 Définitions et généralités	8
2.1.2 Représentation graphique	8
2.1.3 Propriété d'un graphe	9
2.2 Graphe orienté	10
2.2.1 Définitions et généralités	10
2.2.2 Représentation graphique	10
2.2.3 Quelques Propriétés :	11
2.3 Notion de Connexité	11
2.4 Graphe partiel et sous graphe :	12
2.4.1 Définitions :	12
2.4.2 Quelques Types de sous graphes :	12

2.5	Quelques types de graphe	13
2.6	Représentation Structurale d'un graphe	13
2.6.1	Matrice d'adjacence	14
2.6.2	Matrice d'incidence	15
2.6.3	Liste d'adjacence	16
2.7	Les domaines d'application	17
2.7.1	Graphes des réseaux sociaux :	17
2.7.2	Graphes en Bioinformatique :	17
2.7.3	Le Graphe du web :	18
2.8	Conclusion	18
3	Compression de graphe	19
3.1	Compression de données :	19
3.2	Compression appliquée aux graphes :	20
3.2.1	Motivations derrière la compression de graphes :	20
3.2.2	Les types de compression :	20
3.2.3	Les métriques d'évaluation des algorithmes de compression :	21
3.2.4	Classification des méthodes de compression :	22
3.3	Conclusion	22
4	Les méthodes de compression par extraction de motifs	23
4.1	Les méthodes d'extraction de motifs	23
4.2	Compression par extraction de motifs	23
4.3	Conclusion	31
5	Les méthodes de compression k2-Trees	32
6	Conclusion	33

Table des figures

2.1	Exemple de représentation graphique d'un graphe non orienté	9
2.2	Exemple de représentation graphique d'un digraphe.	11
2.3	Graphe orientée G	15
2.4	Matrice d'adjacence du graphe G	15
2.5	Graphe orientée G	16
2.6	Matrice d'incidence du graphe G	16
2.7	Graphe orientée G	16
2.8	Liste d'adjacence du graphe G	16
3.1	Compression sans perte.	20
3.2	Compression avec perte.	21

Liste des tableaux

Chapitre 1

Introduction

Chapitre 2

Théorie des graphes

Pour faciliter la compréhension d'un problème, nous avons tendance à le dessiner ce qui nous amène parfois même à le résoudre. La théorie des graphes est fondée, à l'origine sur ce principe. De nombreuses propriétés et méthodes ont été pensées ou trouvées à partir d'une représentation schématique pour être ensuite formalisées et prouvées.

La théorie des graphes est historiquement un domaine mathématique qui s'est développé au sein des autres disciplines comme la chimie, la biologie, la sociologie et l'industrie. Elle constitue aujourd'hui un corpus de connaissance très important et un instrument efficace pour résoudre une multitude de problèmes.

De manière général le graphe sert à représenter les structures, les connexions entre différents composants, les acheminements possible pour un ensemble complexe composé d'un grand nombre de situations, en exprimant les dépendances et les relations entre ses éléments, (e.g. réseau routier ou ferroviaire, réseau de communication, diagramme d'ordonnancement, ..).

Dans ce chapitre nous présenterons les définitions et les concepts clés relatives aux graphes qui serviront de base pour la suite de notre travail et nous clôturons le chapitre avec quelques domaines d'application de ces derniers.

2.1 Graphe non orienté

2.1.1 Définitions et généralités

Un graphe non orienté G est la donnée d'un couple (V, E) où $V = \{v_1, v_2, \dots, v_n\}$ est un ensemble fini dont les éléments sont appelés sommets ou nœuds (Vertices en anglais) et $E = \{e_1, e_2, \dots, e_m\}$ est un ensemble fini d'arêtes (Edges en anglais). Toute arête e de E correspond à un couple non ordonné de sommets $\{v_i, v_j\} \in E \subset V \times V$ représentant ses extrémités (?) (?).

Soient $e = (v_i, v_j)$ et $e' = (v_k, v_l)$ deux arêtes de E , On dit que :

- v_i et v_j sont les extrémités de e et e est incident en v_i et en v_j (?).
- v_i et v_j sont voisins ou adjacents, car il y'a au moins une arête entre eux dans E (?).
- L'ensemble des sommets adjacents au sommet e est appelé le voisinage de e (?).
- e et e' sont voisins si ils ont une extrémité commune , i.e. : $v_i = v_k$ par exemple (?).
- L'arête e est une boucle si ses extrémités coïncident, i.e. : $v_i = v_j$ (?).
- L'arête e est multiple si elle a plus d'une seule occurrence dans l'ensemble E .

2.1.2 Représentation graphique

Un graphe non orienté G peut être représenté par un dessin sur un plan comme suit (?) :

- Les nœuds de G : $v_i \in V$ sont représentés par des points distincts.
- Les arêtes de G : $e = (v_i, v_j) \in E$ sont représentées par des lignes pas forcément rectilignes qui relient les extrémités de chaque arête e .

Exemple : Soit $g = (V_1, E_1)$ un graphe non orienté tel que : $V_1 = \{1, 2, 3, 5\}$ et $E = \{(1, 2), (1, 4), (2, 2), (2, 3), (2, 5), (3, 4)\}$. La représentation graphique de g est alors donnée par le schéma de la figure 2.1.

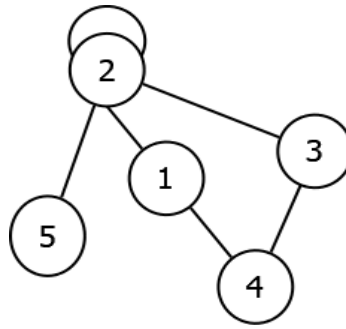


FIGURE 2.1 – Exemple de représentation graphique d'un graphe non orienté

2.1.3 Propriété d'un graphe

- **Ordre d'un graphe** : On appelle ordre d'un graphe le nombre de ses sommets, i.e. $\text{Card}(V)$ (?).
- **Taille d'un graphe** : On appelle taille d'un graphe le nombre de ses arêtes, i.e. $\text{Card}(E)$ (?).
- **Degré d'un graphe** :
 - **Degré d'un sommet** : Le degré d'un sommet noté $d(v_i)$ est le nombre d'arêtes incidentes à ce sommet, sachant qu'une boucle compte pour 2 (?). Dans l'exemple de la figure 2.1, le degré du sommet (1) est : $d(1)=2$.
 - **Degré d'un graphe** : Le degré d'un graphe est le degré maximum de ses sommets, i.e. c'est $\max(d(v_i))$ (?). Dans l'exemple de la figure 2.1, le degré du graphe est $d(2)=5$.
- **Rayon et diamètre d'un graphe** :
 - **Distance** : La distance entre deux sommets v et u est le plus petit nombre d'arêtes qu'on doit parcourir pour aller de v à u ou de u à v (?).
 - **Diamètre d'un graphe** : C'est la plus grande distance entre deux sommets de ce graphe (?).
 - **Rayon d'un graph** : C'est la plus petite distance entre deux sommets de ce graphe.

===== **Rayon d'un graph** : C'est la plus petite distance entre deux sommets de ce graphe (?). »»»> 0e4b8dd6bd0e83a265e861d99710ae5cff85ecfa

2.2 Graphe orienté

2.2.1 Définitions et généralités

Un graphe orienté G est la donnée d'un couple (V, E) où V est un ensemble fini dont les éléments sont appelés les sommets de G et $E \subset V \times V$ est un ensemble de couples ordonnés de sommets dits arcs ou arêtes (?). G est appelé dans ce cas digraphe (directed graphe).

Pour tout arc $e = (v_i, v_j) \in E$:

- v_i est dit extrémité initiale ou origine de e et v_j est l'extrémité finale de e (?).
- v_i est le prédécesseur de v_j et v_j est le successeur de v_i (?).
- les sommets v_i, v_j sont des sommets adjacents (?).
- e est dit sortant en v_i et incident en v_j (?).
- e est appelé boucle si $v_i = v_j$, i.e l'extrémité initiale et finale représente le même sommet (?).

2.2.2 Représentation graphique

Un graphe $G = (V, E)$ peut être projeté sur le plan en représentant :

- dans un premier temps les nœuds $v_i \in V$ par des points disjoints du plan.
- et dans un second temps les arêtes $e = (v_i, v_j) \in E$ par des lignes orientées reliant par des flèches les deux extrémités de e .

Exemple :

Soit $g = (V_1, E_1)$ un digraphe tel que : $V_1 = \{1, 2, 3, 4\}$ et $E_1 = \{(1, 2), (1, 3), (3, 2), (3, 4), (4, 3)\}$.

Le représentation graphique de g est alors donnée par le schéma de la figure ci-dessous.

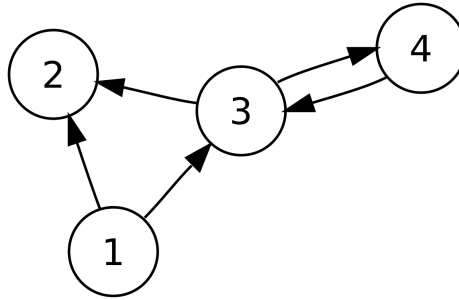


FIGURE 2.2 – Exemple de représentation graphique d'un digraphe.

2.2.3 Quelques Propriétés :

- **Ordre d'un digraphe** : est le nombre de sommets $n = \text{Card}(V)$ (?).
- **taille d'un digraphe** : est le nombre d'arcs $m = \text{Card}(A)$ (?).
- **Degré dans un digraphe** : Le degré d'un sommet $v_i \in V$ dans un digraphe $G = (V, E)$ est donnée par la formule :

$$d(v_i) = d^+(v_i) + d^-(v_i)$$

où $d^+(v_i)$ est le nombre d'arcs sortants au sommet v_i et est appelé degré extérieure et $d^-(v_i)$ représente le nombre d'arcs incidents et est appelé degré intérieur (?).

- **Voisinage dans un digraphe** : Le voisinage d'un sommet $v_i \in V$, noté $V(v_i)$, dans un digraphe $G = (V, E)$ est :

$$V(v_i) = \text{succ}(v_i) \cup \text{pred}(v_i),$$

avec $\text{succ}(v_i)$ qui est l'ensemble des successeurs de v_i et $\text{pred}(v_i)$ qui l'ensemble de ses prédécesseurs (?), i.e le voisinage de v_i est l'ensemble des sommets qui lui sont adjacents.

2.3 Notion de Connexité

Les structures de graphes sont généralement exploitables à travers leurs interrogation qui permet de fournir des réponses aux problèmes modélisés. L'un des informations les

plus importantes dans un graphe est la notion des relations (indirectes ou indirecte) entre deux nœuds ou plus formellement la connexité dans un graphe. Dans cette partie nous allons définir les concepts relatives à cette notion.

- **Chemin (resp. Chaîne) :** est une liste de sommets $S = (v_0, v_1, v_2, \dots, v_k)$ telle qu'il existe un arc (resp. une arête) entre chaque couple de sommets successifs.
- **Cycle (resp. Circuit) :** est un chemin (resp. chaîne) dont le premier et le dernier sommet sont identiques (?).
- **Graphe connexe :** Un graphe non orienté (resp. orienté) est dit connexe (resp. fortement connexe) si pour tout pair de sommets (v_i, v_j) il existe un chemin S les reliant (?).

2.4 Graphe partiel et sous graphe :

La quantité de donnée disponible aujourd'hui et sa croissance de manière exponentiel ont favorisé la décomposition des graphes en des entités plus petites afin de garantir une facilité de compréhension et d'analyse dans le but d'extraire l'information la plus pertinente. Dans cette partie nous allons définir de manière plus formelle ce que ces entités sont ainsi que leurs types.

2.4.1 Définitions :

Soient $G = (V, E)$, $G' = (V', E')$ et $G'' = (V'', E'')$ trois graphes.

- Le graphe G' est appelé graphe partiel de G si : $V' = V$ et $E' \subset E$ (?). En d'autres termes, un graphe partiel est obtenu en supprimant une ou plusieurs arêtes de G .
- Le graphe G'' est dit sous-graphe de G si : $V'' \subset V$ et $E'' \subset E \cap (V'' \times V'')$ (?), i.e un graphe partiel est obtenu en enlevant un ou plusieurs nœuds du graphe initial ainsi que les arêtes dont ils représentent l'une des deux extrémités.

2.4.2 Quelques Types de sous graphes :

- **Une Clique :** est un sous graphe complet de G (?).

- **Bipartie** : G' est un sous graphe bipartie si il existe une partition de V en deux sous ensembles noté V_1 et V_2 , i.e $V' = V_1 \cup V_2$ et $V_1 \cap V_2 = \emptyset$, tel que $E' = V_1 \times V_2$ (?).
- **Étoile** : est un cas particulier de sous graphe bipartie où X est un ensemble contenant le sommet central uniquement et Y contient le reste des nœuds (?) .

2.5 Quelques types de graphe

Avec les avancées technologique au fil du temps, plusieurs types de graphes ont connus le jour. En effet, La complexité et la variété des problèmes scientifiques existants modélisés par ces derniers ont poussé les chercheurs à adapter leurs structure selon le problème auquel ils font face. Durant cette section nous allons définir les principaux types existants.

- **Graphe Complet** : Un graphe $G = (V, E)$ est un graphe complet si tous les sommets $v_i \in V$ sont adjacents (?). Il est souvent noté K_n où $n = \text{card}(V)$ (?).
- **Graphe étiqueté et graphe pondéré** : Un graphe étiqueté $G = (V, E, W)$ est un graphe, qui peut être orienté ou non orienté, dont chacune des arêtes $e_i \in E$ est doté d'une étiquette w_i . Si de plus, w_i est un nombre alors G est dit graphe pondéré (valué) (?).
- **Graphe simple et graphe multiple** : Un graphe $G = (V, E)$ est dit simple si il ne contient pas de boucles et tout pair de sommet est relié par au plus une arête. Dans le cas contraire, G est dit multiple (?).

2.6 Représentation Structurale d'un graphe

Bien que la représentation graphique soit un moyen pratique pour définir un graphe, elle n'est clairement pas adaptée ni au stockage du graphe dans une mémoire, ni à son traitement. Pour cela plusieurs structures de données ont été utilisées pour représenter un graphe, ces structures varient selon l'usage du graphe et la nature des traitements à appliquer. Nous allons présenter dans cette partie les structures les plus utilisées.

Soit un graphe $G(V,E)$ d'ordre n et de taille m dont les sommets v_1, v_2, \dots, v_n et les arêtes (ou arcs) v_1, v_2, \dots, v_m sont ordonnés de 1 à n et de 1 à m respectivement.

2.6.1 Matrice d'adjacence

La matrice d'adjacence de G est une matrice booléenne carrée d'ordre $n : (m_{ij})_{(i,j) \in [0;n]^2}$, dont les lignes (i) et les colonnes (j) représentent les sommets de G , où les entrées (ij) prennent une valeur de "1" s'il existe un arc (une arête dans le cas d'un graph non orienté) allant du sommet i au sommet j et un "0" sinon, e.i (?) (?) :

$$m_{ij} := \begin{cases} 1 & \text{si } (v_i, v_j) \in E \\ 0 & \text{sinon} \end{cases}$$

Dans le cas d'un graphe non orienté, la matrice est symétrique par rapport à la diagonale descendante de gauche à droite . e.i. $m_{ij} = m_{ji}$, dans ce cas le graphe peut être représenté avec la composante triangulaire supérieure de la matrice d'adjacence (?).

Note :

- Cette représentation est valide pour le cas d'un graphe non orienté et orienté.
- Dans le cas d'un graphe pondéré, les "1" sont remplacés par les poids des arêtes (ou arcs) (?).
- Ce mode de représentation engendre des matrices très creuses (comportant beaucoup de zéro) (?).

Exemple : La figure 2.4 représente un exemple de matrice d'adjacence pour le graphe G ci-contre (figure 2.3) :

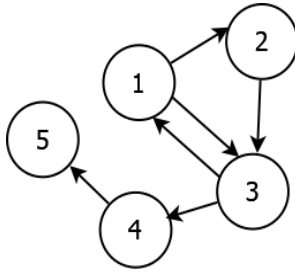


FIGURE 2.3 – Graphe orientée G

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

FIGURE 2.4 – Matrice d'adjacence du graphe G

Place occupé en mémoire : n^2 pour un graphe d'ordre n (?).

2.6.2 Matrice d'incidence

La matrice d'incidence d'un graphe orienté G est une matrice de taille $n \times m$, dont les lignes représentent les sommets ($i \in V$) et les colons représentent les arcs ($j \in E$) et dont les coefficients (m_{ij}) sont dans $\{-1, 0, 1\}$, tel que (?) (?) :

$$m_{ij} := \begin{cases} 1 & \text{si le sommet } i \text{ est l'extrémité final de l'arc } j \\ -1 & \text{si le sommets } i \text{ est l'extrémité initial de l'arc } j \\ 0 & \text{sinon} \end{cases}$$

Pour un graphe non orienté, la coefficients (m_{ij}) de la matrice sont dans $\{0, 1\}$, tel que (?) :

$$m_{ij} := \begin{cases} 1 & \text{si le sommet } i \text{ est une extrémité de l'arête } j \\ 0 & \text{sinon} \end{cases}$$

Exemple : La figure 2.6 représente un exemple de matrice d'incidence pour le graphe G ci-contre (figure 2.5) :

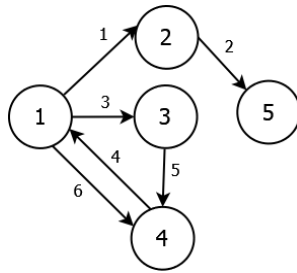


FIGURE 2.5 – Graphe orientée G

$$M = \begin{pmatrix} -1 & 0 & -1 & 1 & 0 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

FIGURE 2.6 – Matrice d'incidence du graphe G

Place occupé en mémoire : $n \times m$

2.6.3 Liste d'adjacence

La liste d'adjacence d'un graphe G est un tableau de n listes, où chaque entrée (i) du tableau correspond à un sommet et comporte la liste T[i] des successeurs (ou prédécesseur) de ce sommet, c'est à dire tous les sommets j tel que $(i,j) \in E$ (?).

Dans le cas d'un graphe non orienté on aura : $j \in \text{la liste } T[i] \iff i \in \text{la liste } T[j]$ (?).

Exemple : La figure 2.8 représente un exemple de matrice d'incidence pour le graphe G ci-contre (figure 2.7) :

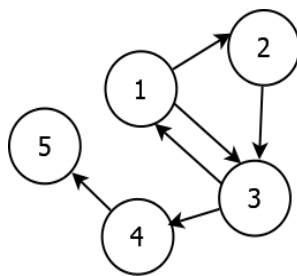


FIGURE 2.7 – Graphe orientée G

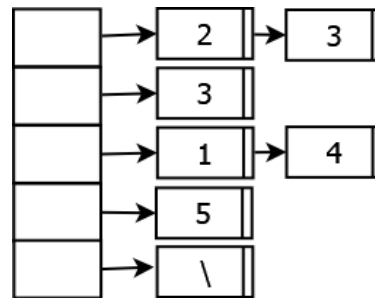


FIGURE 2.8 – Liste d'adjacence du graphe G

Place occupé en mémoire : Dans le cas orienté l'espace est de $n + m$. Dans le cas non orienté, l'espace est de $n + 2m$ car une arête est représenté deux fois.

2.7 Les domaines d'application

La diversité des domaines faisant appel à la modélisation par des graphes ne cesse d'augmenter, allant des réseaux sociaux aux réseaux électriques et réseaux biologiques et arrivant jusqu'aux World Wide Web. Dans cette partie nous allons décrire trois domaines d'application les plus répandus des graphes.

2.7.1 Graphes des réseaux sociaux :

Les réseaux sociaux représentent un lieu d'échange et de rencontre entre individus (entités) et dont l'utilisation est devenue de nos jours une nécessité. Pour représenter les interactions entre ces individus, nous avons généralement besoin de faire recours aux graphes où les sommets sont des individus ou des entités et les interactions entre eux sont représenté par des liens. Vue la diversité des interactions sociales, la modélisation de ces réseaux nécessite différents types de graphes : graphes non orientés pour pour les réseaux sociaux avec des relations non orientées, graphes orientés pour représenter des relations non symétriques comme c'est la cas dans les réseaux de confiance, graphes pondérés pour les réseaux sociaux qui contiennent différents niveaux d'intensités dans les relations, ... etc (?).

2.7.2 Graphes en Bioinformatique :

La bio-informatique est un domaine qui se trouve à l'intersection des deux grands domaines celui de l'informatique et celui de la biologie. Elle a pour but d'exploiter la puissance de calcul des équipements informatiques pour effectuer des traitements sur des données moléculaires massives (?).

Elle est largement utilisée pour l'analyse des séquences d'ADN et des protéines à travers leurs modélisation sous forme de graphe. A titre d'exemple, les graphes non orientés multiples sont un outil modélisation des réseaux d'interaction protéine-protéine (?), le but dans ce cas est donc l'étude du fonctionnement des protéines par rapport à d'autre.

2.7.3 Le Graphe du web :

Le graphe du Web est un graphe orienté dont les sommets sont les pages du web et les arêtes modélise l'existence d'un lien hypertexte dans une page vers une autre (?). Il représente l'un des graphes les plus volumineux : en juillet 2000 déjà, on estimait qu'il contenait environ 2,1 milliards de sommets et 15 milliards d'arêtes avec 7,3 millions de pages ajoutées chaque jour (?). De ce fait, ce graphe a toujours attiré l'attention des chercheurs. En effet, l'étude de ses caractéristiques a donné naissance à plusieurs algorithmes intéressants, notamment l'algorithme PageRank de classement des pages web qui se trouve derrière le moteur de recherche le plus connu de nos jours : Google.

2.8 Conclusion

Dans ce chapitre nous avons présenter les notions et les concepts généraux qui touchent à la théorie de graphes : définitions de graphes, leurs principales propriétés, leurs représentations ainsi que leurs domaines d'application.

Le point important qu'on a put tirer de cette partie est que les graphes sont devenue un moyen crucial et indispensable dans la modélisation des problèmes dans plusieurs domaines. Cependant ils devient de plus en plus complexe et volumineux avec la grande quantités de données disponible de nos jours, ce qui rend leurs stockage, visualisation et traitement difficile. La compression de graphe est nait comme solution à ce problème. Dans le chapitre suivant nous allons présenter la compression de graphe, son rôle et ses différents méthodes.

Chapitre 3

Compression de graphe

3.1 Compression de données :

La compression de donnée est principalement une branche de la théorie de l'information qui traite des techniques et méthodes liées à la minimisation de la quantité de données à transmettre et à stocker. Sa caractéristique de base est de convertir une chaîne de caractères vers un autre jeu de caractères occupant un espace mémoire le plus réduit possible tout en conservant le sens et la pertinence de l'information (?).

Les techniques de compression de données sont principalement motivées par la nécessité d'améliorer l'efficacité du traitement de l'information. En effet, la compression des données en tant que moyen peut rendre l'utilisation des ressources existantes beaucoup plus efficace.

De ce fait, une large gamme d'application usant de ce domaine tel que le domaine des télécommunications et le domaine du multimédia est apparue offrant une panoplie d'algorithmes de compression (?). Sans les techniques de compression, Internet, la télévision numérique, les communications mobiles et les communications vidéo, qui ne cessaient de croître, n'auraient été que des développements théoriques.

3.2 Compression appliquée aux graphes :

3.2.1 Motivations derrière la compression de graphes :

3.2.2 Les types de compression :

La compression de graphe est définie comme l'ensemble des méthodes et techniques permettant de réduire l'espace mémoire occupé par ce derniers sans perte significative d'information. Dès lors, deux approches se présente : la compression avec ou sans perte, que nous allons détaillées dans ce qui suit.

Compression Sans Perte :

Certains domaines d'application de la compression nécessitent un niveau élevé d'exactitude et une restitution exacte, donc une compression sans perte. Dans cette catégorie, le graphe G subi des transformation pour avoir une représentation compacte G' qui lors de la décompression donne exactement G . La figure ci-dessous illustre cette définition.



FIGURE 3.1 – Compression sans perte.

Compression Avec Perte :

Contrairement à la compression sans perte, la compression avec perte permet la suppression permanente de certaines informations jugées inutile (redondantes) pour améliorer la qualité de la compression. En d'autres termes, le graphe G subi des transformations pour avoir une représentation compacte G' qui lors de la décompression donne un graphe G'' probablement différent de G mais l'approximant le plus possible. La figure ci-dessous illustre cette définition.



FIGURE 3.2 – Compression avec perte.

3.2.3 Les métriques d'évaluation des algorithmes de compression :

Devant la panoplie d'algorithmes et de techniques de compression de graphe disponibles dans la littérature, des critères de comparaison et d'évaluation entre ces méthodes doivent être bien définis. Dans cette partie nous présenterons les principales mesures de performances.

Le temps de compression :

C'est une métrique qui donne le temps d'exécution de l'algorithme de compression. Elle est généralement mesurée en secondes (ou ms).

Le ratio de compression :

C'est la mesure la plus courante pour calculer l'efficacité d'un algorithme de compression. Il est défini comme le rapport entre le nombre total de bits requis pour stocker des données non compressées et le nombre total de bits nécessaires pour stocker des données compressées.

$$CR = \frac{\text{No. de bits du graphe originale}}{\text{No. de bits du graphe finale}}$$

Le CR est parfois appelé bit par bit (bpb) et il définit alors comme étant le nombre moyen de bits requis pour stocker les données compressées (?). Dans le cas des algorithmes de compression de graphe on a :

- **Le nombre de bits par nœuds** : représente l'espace mémoire nécessaire pour stocker un nœud (bpn pour bits per node).
- **Le nombre de bits par liens** : représente l'espace mémoire nécessaire pour stocker un lien (bpe pour bits per edge).

Le taux de compression :

Exprimée en pourcentage, cette métrique permet de mesurer la performance de la méthode de compression. Elle peut être exprimée de deux manières différentes :

- **Le taux de compression :** Le rapport entre volume du graphe après compression et le volume initial du graphe.

$$t = \frac{\text{La taille du graphe finale}}{\text{La taille du graphe originale}}$$

- **Le gain d'espace :** Le gain d'espace représente la réduction de la taille du graphe compressé par rapport à la taille du graphe originale.

$$G = 1 - \frac{\text{La taille du graphe finale}}{\text{La taille du graphe originale}}$$

3.2.4 Classification des méthodes de compression :**3.3 Conclusion**

Chapitre 4

Les méthodes de compression par extraction de motifs

4.1 Les méthodes d'extraction de motifs

4.2 Compression par extraction de motifs

Principe général : VOG est une méthode de base sur laquelle s'appuient plusieurs autres méthodes de compression par extraction de motifs, elle décrit le graphe en se basant sur un ensemble appelé vocabulaire composé d'un ensemble de structures communes qui apparaissent souvent dans les graphes du monde réel et qui ont généralement une signification sémantique. De plus VoG utilise le principe de longueur de description minimale (MDL) pour minimiser le coût de la description du graphe en terme de bits.

La problématique traitée par VoG peut être résumée comme suite : étant donné un graphe statique non orienté G , on cherche à trouver un ensemble de sous-graphes chevauchés pour décrire de manière succinct le graphe donné en entrée tout en minimisant le coût du codage en utilisant le principe MDL.

Le principe Minimum Description Length MDL est un concept de la théorie de l'information qui permet de coder un ensemble de données en se basant sur un modèle, tout en minimisant le coût de codage par une fonction objective. Il peut être défini comme suite :

étant donné un ensemble de modèles M , choisir celui qui minimise la fonction suivante : $\min(D, M) = L(M) + L(D | M)$ où $L(M)$ est la longueur, en bits de la description de M et $L(D | M)$ est la longueur, en bits de la description des données en utilisant le modèle M . Pour utiliser le principe MDL dans VoG, il est d'abord essentiel de définir l'ensemble de modèles M , comment un modèle M peut-il décrire un graphe et comment l'encoder. Nous notant aussi que pour une comparaison équitable entre les différents modèles, MDL nécessite que la description soit sans perte.

Soit un graphe non orienté $G(V, E)$, sans boucle avec n nœuds et m arêtes et soit A sa matrice d'adjacence. le résultat de VoG est une liste ordonnée de structures noté M , on note par Ω le vocabulaire composé de six structures qui sont : clique (fc) et quasi-clique (nc), noyau bipartie (cb) et quasi-noyau bipartie (nb), étoile (st) et chaîne (ch). On aura $\Omega = \{ fc, nc, cb, nb, st, ch \}$. Chaque structure $s \in M$ identifie une partie de la matrice d'adjacence A . Nous notons cette partie $\text{area}(s)$. On peut avoir un chevauchement au niveau des nœuds, les liens quand à eux sont servis selon un ordre FIFO et ne peuvent pas être chevauchés, e.i. la première structure $s \in M$ qui décrit l'arête dans A détermine sa valeur.

On note par \mathcal{C}_x l'ensemble de tous les sous-graphes possible de type $x \in \Omega$, et \mathcal{C} l'union de tous ces ensembles, $\mathcal{C} = \cup_x \mathcal{C}_x$. La famille de modèles noté \mathcal{M} représente tous les permutations possibles des éléments de \mathcal{C} . Par MDL, on cherche $M \in \mathcal{M}$ qui minimise le mieux le coût de stockage du modèle et de la matrice d'adjacence.

Pour calculer le coût total de description général, on construit d'abord une approximation \mathbf{M} de la matrice d'adjacence, en utilisant le modèle M : nous considérons de manière itérative chaque structure $s \in M$ et complétons la connectivité de $\text{area}(s)$ dans \mathbf{M} . Comme on a $\mathbf{M} \neq A$ et comme notre méthode est une méthode sans perte, on aura besoin d'une matrice d'erreur E pour reconstituer la matrice d'adjacence tel que $E = A \oplus \mathbf{M}$.

la fonction objective devient donc : $\min(D, M) = L(M) + L(E)$.

Pour l'encodage du modèle, on a pour chaque $M \in \mathcal{M}$:

$$L(M) = L_{\mathbb{N}}(|M|+1) + \log \binom{|M| + |\Omega| - 1}{|\Omega| - 1} + \sum_{s \in M} (-\log Pr(x(s)|M) + L(s))$$

Au début, on calcule le nombre de structures dans le modèle avec $L_{\mathbb{N}}$, et on encode le nombre de structures par type $x \in \Omega$. Ensuite pour chaque structure $s \in M$, on encode son type $x(s)$ avec un code de préfixe optimal et enfin on encode la structure.

Le codage des structures se fait selon leurs type :

Clique : Pour l'encodage d'une clique, on calcule le nombre des nœuds de celle-ci, et on encode leurs ids :

$$L(fc) = L_{\mathbb{N}}(|fc|) + \log \binom{n}{|fc|}$$

Quasi-Clique : Les quasi cliques sont encodées comme des cliques complètes, toute en identifiant les arêtes ajoutées et manquantes en utilisant des codes de préfixe optimaux, on utilise $||nc||$ et $||nc||'$ pour transmettre respectivement le nombre d'arêtes ajoutées et manquantes.

$$L(nc) = L_{\mathbb{N}}(|nc|) + \log \binom{n}{|nc|} + \log(|\text{area}(nc)|) + ||nc||l_1 + ||nc||'l_0$$

Où $l_1 = -\log(||nc||/(||nc||+||nc||'))$ et analogue à l_0 sont les longueurs des codes de préfixe optimaux des arêtes ajoutées et manquantes.

Noyau bipartie : notant par A et B les deux ensemble du noyau bipartie, On encode leurs tailles, ainsi que les ids de leurs sommets :

$$L(fb) = L_{\mathbb{N}}(|A|) + L_{\mathbb{N}}(|B|) + \log \binom{n}{|A|} + \log \binom{n - |A|}{|B|}$$

Quasi-Noyau bipartie : Comme les quasi-cliques, les noyau bipartie sont codé comme suit :

$$L(nb) = L_{\mathbb{N}}(|A|) + L_{\mathbb{N}}(|B|) + \log \binom{n}{|A|} + \log \binom{n - |A|}{|B|} + \log(|\text{area}(nb)|) + ||nb||l_1 + ||nb||'l_0$$

Étoile : L'étoile est un cas particulier des noyau bipartie où l'un des ensembles est constituer d'un seule élément (appelé hub) et l'autre ensemble est constituer des sommets restants (appelé spokes). Le codage est calculer comme suit, d'abord on calcule la nombre de spokes de l'étoile, ensuite on identifie le hub parmi les n sommets et les spokes parmi les $n-1$ restants.

$$L(st) = L_{\mathbb{N}}(|st|-1) + \log n + \log \binom{n-1}{|st|-1}$$

Chaîne : On calcule d'abord le nombre d'éléments de la chaîne, ensuite on encode les ids des nœuds selon leurs ordre dans la chaîne :

$$L(ch) = L_{\mathbb{N}}(|ch| - 1) + \sum_{i=0}^{|ch|} (n - i)$$

Matrice d'erreur : la matrice d'erreur E est encoder sur deux parties E^+ et E^- . E^+ correspond a la partie de A que M modélise en rajoutant des liens non existants. contrairement à E^- qui représente les partie de A que M ne modélise pas. Notons que les quasi-clique et les quasi-noyau bipartie ne sont pas inclut dans la matrice d'erreur puisque ils sont encodés exactement donc on les ignorent. Le codage de E^+ et E^- est similaire a celui des quasi-clique, on a :

$$\begin{aligned} L(E^+) &= \log(|\text{area}(E^+)|) + \|E^+\|_1 + \|E^+\|'l_0 \\ L(E^-) &= \log(|\text{area}(E^-)|) + \|E^-\|_1 + \|E^-\|'l_0 \end{aligned}$$

Algorithme de VoG : Pour la recherche du meilleur modèle $M \in \mathcal{M}$, VoG procède sur trois étapes :

1. Génération des sous-structures : Dans cette phase, Les méthodes de détection de communautés et de clustering sont utilisé pour décomposer le graphe en sous-graphes pas forcément disjoints. La méthode de décomposition utilisé dans VOG est SlashBurn.
2. Étiquetage des sous-graphes : L'algorithme cherche pour chaque sous-graphe généré dans l'étape précédente la structure $x \in \Omega$ qui le décrit le mieux, en tolérant un certain seuil d'erreur.

a Étiquetage des structures parfaites : Tout d'abord, le sous-graphe est testé par rapport aux structures complètes du vocabulaire (e.i. clique, étoile, chaîne et noyau bipartie) pour une similarité sans erreur. Le test des cliques et des chaînes est basé sur la distribution des degrés. Plus précisément, si tous les sommets d'un sous graphe d'ordre n ont un degré égal à $n-1$, il s'agit alors d'une clique. De même si tous les sommets ont un degré de 2 sauf deux sommets ayant le degré 1, le sous-graphe est une chaîne. D'un autre côté, le sous-graphe est considéré comme noyau bipartie si les amplitudes de ses valeurs propres maximales et minimales sont égales, pour pouvoir identifier les sommets de chaque ensemble du noyau nous utilisons le parcours BFS avec la coloration des sommets. Quant à l'étoile, elle est considérée comme un cas particulier d'un noyau bipartie, il suffit donc que l'un des ensembles soit composé d'un seul sommet.

b Étiquetage des structures approximative : Si le sous graphe ne correspond pas à une structure complète, on cherche la structure qui l'approxime mieux en terme de MDL. Pour ce faire, on le représente sous forme de chaque type $x \in \Omega$ et on choisit le type avec la description minimal.

Remarque : Pour les structures parfaites (clique, étoile, chaîne et noyau bipartie), en plus du coût d'encodage de la structure, on rajoute le coût de l'erreur local c'est à dire, $L(x^*) + L(E_{x^*}^+) + L(E_{x^*}^-)$ où $L(x^*)$ est la description de la structure, $L(E_{x^*}^+)$ et $L(E_{x^*}^-)$ sont les arêtes incorrectement modélisées et les arêtes non modélisées respectivement dans $\text{area}(x^*)$.

Après avoir représenté le sous graphe sous forme d'une structure, on l'ajoute à l'ensemble des structures candidates \mathcal{C} , en l'associant à son coût.

3. Assemblage du modèle : Dans cette dernière étape, une sélection d'un ensemble de structures parmi ceux de \mathcal{C} est réalisée, des heuristiques de sélection sont utilisées car le nombre de permutations est très grand ce qui implique des calculs exhaustifs. Les heuristiques permettent d'avoir des résultats approximatifs et rapides, parmi les heuristiques utilisés dans VOG on trouve :

- PLAIN : Cette heuristique retourne tout les structures candidates. e.i. $M = \mathcal{C}$.
- TOP-K : Cette heuristique sélectionne les k meilleurs candidats en fonction de leurs gain en bits.
- GREEDY'N FORGET(GNF) : Parcours structure par structure dans l'ensemble \mathcal{C} ordonnés par leurs qualité (gain en bits), ajoute la structure au modèle tant que elle n'augmente pas le cout total de la représentation, sinon l'ignore.

Ci-dessous le pseudo algorithme de VOG :

- 1: **Entré** : Graphe G .
- 2: **Étape 1** : Génération des sous-graphe en utilisant une méthode de décomposition.
- 3: **Étape 2** : Étiquetage des sous graphe en choisissant la structure $x \in \Omega$ qui représente chaque sous- graphe avec le moindre coût.
- 4: **Étape 3** : Assemblage des sous graphes en utilisant des heuristiques pour sélectionner un sous-ensemble non-redondant à partir des structure candidate de l'étape 2.
- 5: **Sortie** : Modèle M et son cout d'encodage.

Algorithm 1: Pseudo Algorithme VOG

Comme nous l'avons déjà précisé, VOG formule le problème de compression de graphe en tant que problème d'optimisation basé sur la théorie de l'information, l'objectif étant de rechercher les structures qui minimisent la longueur de description globale du graphe. Un élément clé de VoG est la méthode de décomposition utilisée qui peut donner en sortie des sous-graphes ayant des nœuds et/ou des arêtes en commun et dont VoG(?) ne suppose que le premiers cas.

En partant de ce constat, les auteurs de (?) propose VoG-overlapp, une extension de VoG prenant en compte les chevauchement des structures sous forme d'une étude de l'effet de diverses méthodes de décomposition sur la qualité de la compression, à savoir la méthodes SlushBurn, Louvain, Spectral clustering, Metis et KCBC qui est une nouvelle méthode proposée dans cette article et dont la description et fournie dans l'(Algorithme 1).

L'idée de base de VoG-overlapp est d'inclure une pénalité pour les chevauchements importants dans la fonction objective ce qui oriente le processus de sélection des structures vers la sortie souhaitée. Elle devient alors :

$$\min L(G, M) = \min \{L(M) + L(E) + \mathbf{L}(\mathbf{O})\}$$

Le principe de calcul de $L(M)$ et $L(E)$ demeurent le même avec \mathbf{O} , une matrice cumulant le nombre de fois que chacune des arêtes a été couverte par le modèle. Le cout du codage de la matrice des chevauchement est donné par la formule (4.1).

$$L(\mathbf{O}) = \log(|\mathbf{O}|) + \|\mathbf{O}\| l_1 + \|\mathbf{O}\|' l_0 + \sum_{o \in \varepsilon(\mathbf{O})} L_N(|o|) \quad (4.1)$$

Où :

- $|\mathbf{O}|$ est le nombre d'arêtes (distinctes) qui se répète dans le modèles \mathbf{M} .
- $\|\mathbf{O}\|$ et $\|\mathbf{O}\|'$ représentent respectivement le nombre des arêtes présentes et manquantes dans \mathbf{O} .
- $l_1 = -\log(\frac{\|\mathbf{O}\|}{\|\mathbf{O}\| + \|\mathbf{O}\|'})$, de manière analogue l_0 , sont les longueurs des codes de préfixe optimaux pour les arêtes actuelles et manquantes, respectivement.
- $\varepsilon(\mathbf{O})$ est l'ensemble des entrées non nulles dans la matrice \mathbf{O} .

Durant la même année, (?) ont proposé une autre variation de VoG, TimeCrunch, pour le cas des graphes simples (sans boucles) non orientés dynamiques représentés par un ensemble de graphes associés chacun à timestamp. En d'autres termes, ils considèrent les graphes $G = \bigcup_{t_i} G_{t_i}(V, E_{t_i})$ $1 \preceq i \preceq t$ où $G_{t_i} = G$ a l'instant t_i .

Un nouveau vocabulaire est proposé pour décrire proprement l'évolution des sous-structures dans le temps. En effet, ils partent du même vocabulaire de structures statiques $\Omega = \{st(etoile), fc(clique), nc(quasi-clique), bc(bipartie), nb(quasi-bipartie), ch(chaine)\}$ dont ils affectent une signature temporelle $\delta \in \Delta$ où : $\Delta = \{o(oneshot), r(ranged), p(periodique), f(flickering), c(constante)\}$.

Comme les éléments du modèle sont modifiés, son cout est alors aussi modifié pour inclure pour chaque structure s non seulement sa connectivité $c(s)$ correspondant aux arêtes des zones induites par s mais aussi sa présence temporelle $u(s)$ correspondant aux timestamps dans lesquels s apparait dans le graphe G .

$$L(M) = L_N(|M| + 1) + \log\left(\frac{|M|+|\Phi|-1}{|\Phi|-1}\right) + \sum_{s \in M} (\log P(v(s)|M) + L(c(s)) + \mathbf{L}(\mathbf{u}(s)))$$

Le cout de l'encodage de la présence temporelle diffèrent selon ses caractéristiques. Nous présenterons dans ce qui suit la formule correspondant à chaque signature.

- **Oneshot** : cette signature décrit les sous-structure qui apparaissent dans un seul timestamp ,i.e $|u(s)| = 1$. Donc le cout de l'encodage se réduit aux nombre de bits nécessaires pour sauvegarder le timestamp : $L(u(s)) = \log(t)$.
- **Ranged** : dans ce cas la sous-structure apparait dans tous les graphes se trouvant entre deux timestamps t_1 et t_2 . Le cout englobe le nombre de timestamps dans lesquels elle apparait ainsi que les identifiants des deux timestamp t_{debut} et t_{fin} : $L(u(s)) = L_N(|u(s)|) + \log\left(\frac{t}{2}\right)$.
- **Periodic** : cette catégorie est une extension de la précédente avec les timestamps qui sont éloigné de plus d'un pas d'où : $L(p) = L(r)$.
En effet, la périodicité peut être déduite des marqueurs début et de fin ainsi que du nombre de pas de temps $|u(s)|$, permettant ainsi de reconstruire $u(s)$
- **Flickering** : ce type décrit les structures qui apparaissent dans n timesteps de manière aléatoire. Le cout doit englober donc le nombre de timesteps ainsi que leurs identifiants d'où : $L(u(s)) = L_N(|u(s)|) + \log\left(\frac{t}{|u(s)|}\right)$.
- **Constant** : dans ce cas la sous-structure apparait dans tout les timesteps et donc elle ne dépend pas du temps d'où $L(c)=0$.

Nous notons que décrire $u(s)$ est encore un autre problème de sélection de modèle pour lequel nous pouvons tirer parti de MDL. En effet juste comme pour le codage de la connectivité, il peut ne pas être précis avec une signature temporelle donnée. Toutefois, toute approximation entraînera des coûts supplémentaires pour l'encodage de l'erreur qui

englobent l'erreur de l'encodage de la connectivité ainsi que l'erreur de l'encodage de la signature temporelle.

4.3 Conclusion

Chapitre 5

Les méthodes de compression

k2-Trees

Chapitre 6

Conclusion

Bibliographie

- (2012). *Quelques rappels sur la théorie des graphes*. IUT Lyon Informatique.
- Brisaboa, N. R., Ladra, S., and Navarro, G. (2009). k 2-trees for compact web graph representation. In *International Symposium on String Processing and Information Retrieval*, pages 18–30. Springer.
- Fages, J.-G. (2014). *Exploitation de structures de graphe en programmation par contraintes*. PhD thesis, Ecole des Mines de Nantes.
- Guillaume, J.-L. and Latapy, M. (2002). The web graph : an overview. In *Actes d'ALGOTEL'02 (Quatrièmes Rencontres Francophones sur les aspects Algorithmiques des Télécommunications)*.
- Hennecart, F., Bretto, A., and Faisant, A. (2012). *Eléments de théorie des graphes*.
- Jean-Charles Régin, A. M. (2016). *Théorie des graphes*. Technical report.
- Koutra, D., Kang, U., Vreeken, J., and Faloutsos, C. (2015). Summarizing and understanding large graphs. *Statistical Analysis and Data Mining : The ASA Data Science Journal*, 8(3) :183–202.
- Lehman, E., Leighton, F. T., and Meyer, A. R. (2010). *Mathematics for computer science*. Technical report, Technical report, 2006. Lecture notes.
- Lelewer, D. A. and Hirschberg, D. S. (1987). Data compression. *ACM Computing Surveys (CSUR)*, 19(3) :261–296.

- Lemmouchi, S. (2012). *Etude de la robustesse des graphes sociaux émergents*. PhD thesis, Université Claude Bernard-Lyon I.
- Liu, Y., Shah, N., and Koutra, D. (2015). An empirical comparison of the summarization power of graph clustering methods. *arXiv preprint arXiv :1511.06820*.
- Lopez, P. (2003). Cours de graphes.
- Müller, D. (2012). *Introduction à la théorie des graphes*. Commission romande de mathématique (CRM).
- Parlebas, P. (1972). Centralité et compacité d'un graphe. *Mathématiques et sciences humaines*, 39 :5–26.
- Pellegrini, M., Haynor, D., and Johnson, J. M. (2004). Protein interaction networks. *Expert review of proteomics*, 1(2) :239–249.
- Rigo, M. (2010). *Théorie des graphes*. Université de Liège, Faculté des sciences Département de mathématiques.
- Roux, P. (2014). *Théorie des graphes*.
- SABLIK, M. (2018). Graphe et langage.
- Sethi, G., Shaw, S., Vinutha, K., and Chakravorty, C. (2014). Data compression techniques. *International Journal of Computer Science and Information Technologies*, 5(4) :5584–6.
- Shah, N., Koutra, D., Zou, T., Gallagher, B., and Faloutsos, C. (2015). Timecrunch : Interpretable dynamic graph summarization. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1055–1064. ACM.
- Uthayakumar, J., Vengattaraman, T., and Dhavachelvan, P. (2018). A survey on data compression techniques : From the perspective of data quality, coding schemes, data

type and applications. *Journal of King Saud University-Computer and Information Sciences*.