République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي

ESI ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الإعلام الآلي سابقا)
Ecole nationale Supérieure d'Informatique
ex. INI (Institut National de formation en Informatique)

Mémoire

Pour Obtention du diplôme de Master En Informatique

Option : Système Informatique (SIQ)

Étude et classification des méthodes de compression de graphe par extraction de motifs et k2-trees

---

Réaliser par :

Mlle. Hafsa Bousbiat
eh_bousbiat@esi.dz
ESI

Mlle. Sana Ihadadene
es_ihadadene@esi.dz
ESI

Encadrer par :

Dr. Karima Amrouche
k_amrouche@esi.dz
ESI

Dr. Hamida Seba
hamida.seba@univ-lyon1.fr
Université de Lyon

Octobre 2018
Année Universitaire : 2018-2019

Remerciement

Nous tenons d'abord à exprimer notre gratitude à l'égard de nos deux encadrantes, Madame Seba Hamida, Madame Amrouche Karima pour leurs précieuse aide, leurs judicieux conseils et pour le temps qu'elles nous ont consacré tout au long de ce travail.

Nous remercions également toute l'équipe pédagogique de l'École Nationale Supérieure d'Informatique, et à leur tète nos enseignants, pour la richesse et la qualité de la formation qu'ils nous ont offert tout au long de notre cursus.

Nous tenons aussi à remercier les membres du jury qui nous ont fait l'honneur d'accepter de juger cet humble travail.

Enfin, nous adressons nos plus sincères remerciements à tous nos proches et amis, qui nous ont toujours encouragés au cours de la réalisation de ce mémoire.

Merci à tous et à toutes ...

Résumé

Nous vivons dans un monde où la quantité d'informations ne cesse d'augmenter et dont la bonne gestion implique l'utilisation des graphes qui se sont répandus dans différents domaines allant des réseaux sociaux et de communication jusqu'aux domaines de la chimie et de la biologie. Cette abondance de données générées fait appel à une technique aussi vieille que la discipline de traitement de données mais qui connait de nouveaux défis aujourd'hui : la compression. La compression de graphes est un domaine dans lequel le graphe initial subit des transformations pour en obtenir une version plus réduite et compacte permettant, dans la majorité des cas, d'effectuer les traitements dans un temps nettement meilleur.

Dans ce travail, nous étudierons les différentes méthodes de compression existantes dans la littérature dans le but de compléter et d'affiner davantage une classification entamée dans un travail de Master précédent. Nous mettrons l'accent sur les méthodes de compression basées sur l'extraction de motifs ainsi que les méthodes basées sur les k^2-trees. Par conséquent, nous proposons d'établir une synthèse bibliographique sur ces deux classes et nous établirons une étude comparative entre les méthodes de ces deux classes selon divers critères.

Mots Clés : *Compression de graphes, Big Data, Extraction de motifs, K2-trees, Graphe du Web.*

Abstract

We live in a world where the amount of information is constantly increasing and whose good management involves the use of graphs that have spread in different fields from social and communication networks to the fields of chemistry and biology. This abundance of generated data calls for a technique that is as old as the discipline of data processing but which is facing new challenges today : the compression. Graph compression is a field in which the initial graph undergoes transformations in order to obtain a smaller and more compact version allowing, in the majority of cases, to perform the treatments in a much better time.

In this work, we will study the different methods of compression existing in the literature in order to complete and further refine a classification started in a previous Master's work. We will focus on compression methods based on pattern extraction as well as k^2 -trees methods. Therefore, we propose to establish a bibliographic synthesis on these two classes and we will establish a comparative study between the methods of these two classes according to various criteria.

Key words : *Graph compression, Big Data, Pattern extraction, K2-trees, Web graph.*

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Avec l'énorme quantité de données produites par les activités humaines de nos jours, le problème de données massives (Big data) est devenu un enjeu essentiel. Un des outils les plus efficaces pour traiter ces données est l'utilisation de graphes. Les graphes sont des outils de modélisation utilisés dans beaucoup de domaines pour la représentation des données : réseaux sociaux et de communication (entités reliées entre elles par des liens physiques ou communautaires), chimie (relations entre les atomes), biologie (interactions entre protéines par exemple), etc.

Il y a plusieurs solutions pour contrer le volume de données mais nous nous intéressons à une solution aussi vieille que la discipline de traitement de données mais qui connait de nouveaux défis de nos jours : la compression.

La compression de graphes est un domaine dans lequel le graphe initial subit des transformations pour en obtenir une version plus réduite. Différentes techniques permettent cette compression, avec ou sans perte d'information, et génèrent de nouveaux graphes sur lesquels il est intéressant d'effectuer les différents traitements.

Deux types de méthodes de compression de graphes se sont distingués parmi tous les autres types de méthodes : les k2-trees et les méthodes de compression par extraction de motifs. En effet, elles permettent de trouver dans la majorité des cas un bon compromis entre l'espace mémoire et les temps de traitement. Nous présenterons dans ce travail une synthèse bibliographique sur les différentes méthodes de compression dérivant de ces deux grandes classes.

Nous avons hiérarchisé notre mémoire en trois grands chapitres. Le premier est une introduction au domaine de la théorie des graphes incluant ainsi les notion fondamentales de ce domaine, les types de graphes, leurs représentations structurelles et leurs domaines d'application. Dans le second chapitre, nous introduisons les définitions de base du domaine de compression de données appliqué aux graphes à savoir : les motivation, les types de compression et les métriques d'évaluation. Enfin, nous présenterons les classifications des méthodes existantes de compression de graphe ainsi que la classification que nous proposons. Nous mettrons l'accent après sur les deux classes de méthodes qui forment la problématique de notre sujet de Master : la compression par extraction de motifs et la compression par les k2-trees. Dans le troisième chapitre nous présenterons une nouvelle méthode de compression que nous proposons.

Chapitre 1

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 générale, le graphe sert à représenter les structures, les connexions entre différents composants, les acheminements possibles 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 notions et les concepts clés relatives aux graphes qui serviront de base pour la suite de notre travaille, à savoir : la définition d'un graphe, ses types et sa représentation structurelle. Nous clôturons le chapitres avec quelques domaines d'application des graphes.

1.1 Graphe non orienté

1.1.1 Définitions et généralités

Un graph non orienté G est la donnée d'un couple (V , E) où V = $\{v_1, v_2, ..., v_n\}$ est un ensemble fini dont les éléments sont appelés sommets ou nœuds (Vertices en anglais) et E=$\{ e_1, e_2, \ldots, e_m \}$ est un ensemble fini d'arêtes (Edges en anglais). Toute arête e de E correspond à un couple non ordonné de sommets $\{ v_i, v_j \} \in \mathrm{E} \subset V \times V$ représentants ses extrémités (Müller, 2012) (Fages, 2014).
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 (Hennecart et al., 2012).

— v_i et v_j sont voisins ou adjacents, car il y a au moins une arête entre eux dans E (IUT, 2012).

— L'ensemble des sommets adjacents au sommet e est appelé le voisinage de e (Müller, 2012).

— e et e' sont voisins si ils ont une extrémité commune , i.e. : $v_i = v_k$ par exemple (Lopez, 2003).

— L'arête e est une boucle si ses extrémités coïncident, i.e. : $v_i = v_j$ (IUT, 2012).

— L'arête e est multiple si elle a plus d'une seule occurrence dans l'ensemble E.

1.1.2 Représentation graphique

Un graph non orienté G peut être représenter par un dessin sur un plan comme suit (Müller, 2012) :

— Les nœuds de G : $v_i \in$ V sont représentés par des points distincts.

— Les arêtes de G : $e = (v_i,v_i) \in$ E sont représentés par des lignes pas forcement rectilignes qui relient les extrémités de chaque arête e.

Exemple : Soit g=(V1 , E1) un graphe non orienté tel que : V1={ 1,2,3,5 } et E1={ (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 1.1.

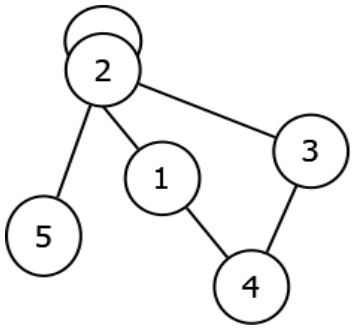

FIGURE 1.1 – Exemple de représentation graphique d'un graphe non orienté

1.1.3 Propriétés d'un graphe

- **Ordre d'un graphe :** On appel ordre d'un graphe le nombre de ses sommets.i.e. Card(V) (Roux, 2014).
- **Taille d'un graphe :** On appel taille d'un graphe le nombre de ses arêtes.i.e. Card(E) (Roux, 2014).
- **Degré dans 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 deux (Müller, 2012). Dans l'exemple de la figure 1.1, 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($d(v_i)$) (Müller, 2012). Dans l'exemple de la figure 1.1, le degré du graphe est $d(2)$=5.

◦ **Rayon et diamètre dans 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 (Müller, 2012).
- **Diamètre d'un graphe :** C'est la plus grande distance entre deux sommets de ce graphe (Müller, 2012).
- **Rayon d'un graph :** C'est la plus petite distance entre deux sommets de ce graphe (Parlebas, 1972).

1.2 Graphe orienté

1.2.1 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 x V est un ensemble de couples ordonnés de sommets dits arcs ou arêtes (Müller, 2012). G est appelé dans ce cas digraphe (directed graph).
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 (Müller, 2012).

— v_i est le prédécesseur de v_j et v_j est le successeur de v_i (IUT, 2012).

— les sommets v_i , v_j sont des sommets adjacents (Jean-Charles Régin, 2016).

— e est dit sortant en v_i et incident en v_j (Jean-Charles Régin, 2016).

— e est appelé boucle si $v_i = v_j$, i.e l'extrémité initiale et finale représente le même sommet (IUT, 2012).

1.2.2 Représentation graphique

Un graphe G = (V , E) peut être projeter 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 donné par le schéma de la figure 1.2.

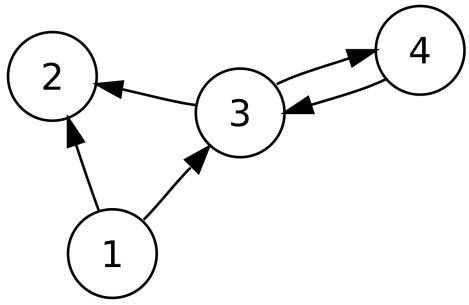

FIGURE 1.2 – Exemple de représentation graphique d'un digraphe.

1.2.3 Quelques Propriétés :

- **Ordre d'un digraphe :** est le nombre de sommets n = Card(V) (Roux, 2014).
- **taille d'un digraphe :** est le nombre d'arcs m = Card(A) (Roux, 2014).
- **Degré dans un digraphe :** Le degré d'un sommet $v_i \in$ V dans un digraphe G=(V , E) est donnée par la formule :

  $$\mathrm{d}(v_i) = d^+(v_i) + d^-(v_i$$

  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 (Müller, 2012).
- **Voisinage dans un digraphe :** Le voisinage d'un sommet $v_i \in$ V, noté V(v_i), dans un digraphe G = (V , E) est :

  $$\mathrm{V}(v_i) = \mathrm{succ}(v_i) \bigcup \mathrm{pred}(v_i),$$

  où succ(v_i) est l'ensemble des successeurs de v_i et pred(v_i) est l'ensemble de ses prédécesseurs (Rigo, 2010), i.e le voisinage de v_i est l'ensemble des sommets qui lui sont adjacents.

1.3 Notion de Connexité

Les graphes sont généralement exploitables à travers leurs interrogation qui permet de fournir des réponses aux problèmes modélisés. L'une des informations les plus importantes dans un graphe est la notion des relations (directes ou indirectes) 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. Chaine) :** est une liste de sommets S= $(v_0, v_1, v_2, ..., v_k)$ telle qu'il existe un arc (resp. une arête) entre chaque couple de sommets successifs.
- **Cycle (resp. Circuit) :** est un chemin (resp. chaine) dont le premier et le dernier sommet sont identiques (Roux, 2014).

- **Graphe connexe :** Un graphe non orienté (resp. orienté) est dit connexe (resp. fortement connexe) si pour toute paire de sommets (v_i, v_j), il existe un chemin S les reliant (Müller, 2012).

1.4 Graphe partiel et sous graphe :

La quantité de données 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.

1.4.1 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 (Roux, 2014). 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" x V") (Rigo, 2010), 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.

1.4.2 Quelques Types de sous graphes :

- **Une Clique :** est un sous graphe complet de G (Rigo, 2010).
- **Bipartie :** G' est un sous graphe bipartie si il existe une partition de V' en deux sous ensembles notés $V_1 et V_2$, i.e V' = $V_1 \cup V_2$ et $V_1 \cap V_2 = \phi$, tel que E' = V_1 x V_2 (Rigo, 2010).
- **Étoile :** est un cas particulier de sous graphe bipartie où V_1 est un ensemble contenant le sommet central uniquement et V_2 contient le reste des nœuds (Koutra et al., 2015) .

1.5 Quelques types de graphe

Avec les avancées technologiques au fil du temps, plusieurs types de graphes ont connus le jours. 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 (Jean-Charles Régin, 2016). Il est souvent noté K_n où n = card(V) (Roux, 2014).

- **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) (Roux, 2014).
- **Graphe simple et graphe multiple :** Un graphe G = (V , E) est dit simple si il ne contient pas de boucles et toute paire de sommets est reliée par au plus une arête. Dans le cas contraire, G est dit multiple (IUT, 2012).

1.6 Représentation Structurelle 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 appliqués. 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,..., v_n et les arêtes (ou arcs) e_1, e_2,..., e_m sont ordonnés de 1 à n et de 1 à m respectivement.

1.6.1 Matrice d'adjacence

La matrice d'adjacence de G est une matrice booléenne carré d'ordre n : $(m_{ij})_{(i,j)\in[0;n]^2}$, dont les lignes (i) et les colons (j) représentent les sommets de G. 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, i.e (Lehman et al., 2010) (SABLIK, 2018) (IUT, 2012) :

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

Dans le cas d'un graphe non orienté, la matrice est symétrique par rapport à la première diagonale, i.e. $m_{ij} = m_{ji}$, dans ce cas le graphe peut être représenter avec la composante triangulaire supérieure de la matrice d'adjacence (Müller, 2012).

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) (Lopez, 2003).

— Ce mode de représentation engendre des matrices très creuses (comprenant beaucoup de zero) (Hennecart et al., 2012).

Exemple : La figure 1.4 représente un exemple de matrice d'adjacence pour le graphe G ci-contre (figure 1.3) :

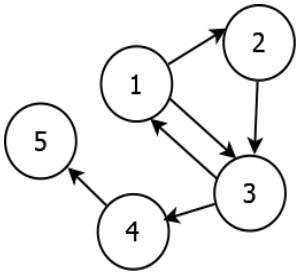

FIGURE 1.3 – Graphe orienté G

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

FIGURE 1.4 – Matrice d'adjacence du graphe G

Place occupé en mémoire : n^2 pour un graphe d'ordre n (Lopez, 2003).

1.6.2 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 colonnes représentent les arcs (j $\in$ E) et dont les coefficients (m_{ij}) sont dans {-1, 0, 1}, tel que (Hennecart et al., 2012) (SABLIK, 2018) :

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

Pour un graphe non orienté, les coefficients (m_{ij}) de la matrice sont dans {0, 1}, tel que (Hennecart et al., 2012) :

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

Exemple : La figure 1.6 représente un exemple d'une matrice d'incidence pour le graphe G ci-contre (figure 1.5) :

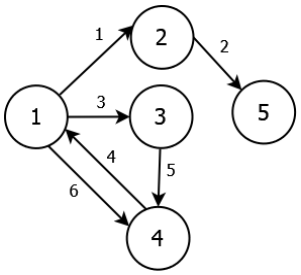

FIGURE 1.5 – Graphe orienté G

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

FIGURE 1.6 – Matrice d'incidence du graphe G

Place occupé en mémoire : $n \times m$

1.6.3 Liste d'adjacence

La liste d'adjacence d'un graphe G est un tableau de n listes, où chaque entrée (i) du tableau correspond a 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) ∈ E (SABLIK, 2018).
Dans le cas d'un graphe non orienté on aura : j ∈ la liste T[i] $\iff$ i ∈ la liste T[j] (IUT, 2012).

Exemple : La figure 1.8 représente un exemple d'une liste d'adjacence pour le graphe G ci-contre (figure 1.7) :

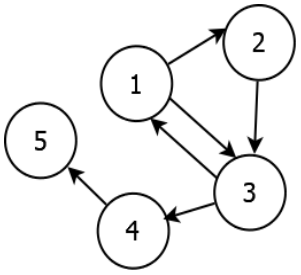

FIGURE 1.7 – Graphe orienté G

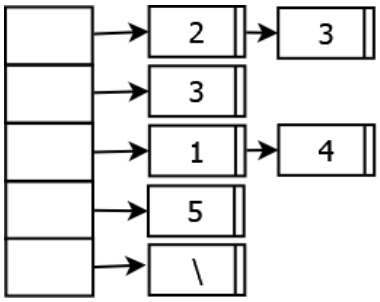

FIGURE 1.8 – Liste d'adjacence du graphe G

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.

1.7 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.

1.7.1 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 représentées par des liens. Vue la diversité des interactions sociales, la modélisation de ces réseaux nécessite différents types de graphes : graphes non orientés pour les réseaux sociaux avec des relations symétriques, 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 (Lemmouchi, 2012).

1.7.2 Graphes en Bioinformatique :

La bio-informatique est un domaine qui se trouve à l'intersection de deux grands domaines celui de l'informatique et celui de la biologie. Elle a pour but d'exploiter la puissance de calcule des équipements informatiques pour effectuer des traitements sur des données moléculaires massives (Pellegrini et al., 2004).

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 de modélisation des réseaux d'interaction protéine-protéine (Pellegrini et al., 2004), le but dans ce cas est donc l'étude du fonctionnement des protéines par rapport à d'autre.

1.7.3 Le Graphe du web :

Le graphe du Web est un graphe orienté dont les sommets sont les pages du web et les arêtes modélisent l'existence d'un lien hypertexte dans une page vers une autre (Brisaboa et al., 2009). Il représente l'un des graphes les plus volumineux : en juillet 2000 déja, 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 (Guillaume and Latapy, 2002). 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.

1.8 Conclusion

Dans ce chapitre nous avons présenter les notions et les concepts généraux qui touchent à la théorie des graphes : définitions des 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 devenus un moyen crucial et indispensable dans la modélisation des problèmes dans plusieurs domaines. Cependant ils deviennent de plus en plus complexes et volumineux avec la grande quantité de données disponible de nos jours, ce qui rend leurs stockage, visualisation et traitement difficile. La compression de graphe est naît comme solution à ce problème. Dans le chapitre suivant nous allons présenter la compression de graphes, son rôle et ses différents méthodes.

Chapitre 2

Compression de graphe

2.1 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 (Lelewer and Hirschberg, 1987).

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 (Sethi et al., 2014). 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.

2.2 Compression appliquée aux graphes :

La compression de graphes a été proposée comme solution pour le traitement et le stockage des graphes volumineux, elle permet de transformer un grand graphe en un autre plus petit tout en préservant ses propriétés générales et ses composants les plus importants. Les principaux avantages de la compression sont (Liu et al., 2018a) :

— Réduction de la taille des données et de l'espace de stockage : De nos jours, les graphes représentant les bases de donnés, les réseaux sociaux et tous types de données numériques accroissent d'une manière exponentielle, ce qui rend leur stockage difficile et couteux en terme d'espace mémoire. Les techniques de compression produisent des graphes plus petits qui nécessitent moins d'espace que leurs graphes d'origines. Cela permet aussi de décroitre le nombre d'opérations d'E/S, de réduire

les communications entre nœuds dans un environnement distribué et de charger le graphe en mémoire centrale.

— Exécution rapide des algorithmes de traitement et des requêtes sur les graphe : l'exécution des différents algorithmes de traitement sur des graphes volumineux peut avérer couteuse en terme de temps et peut ne pas donner les résultats attendues. La compression permet d'obtenir de petits graphes qui peuvent être traiter, analyser et interroger plus efficacement et dans un temps raisonnable.

— Facilité d'analyse et visualisation du graphe : Les techniques de compression permettent de représenter les données et les structures des graphes massives d'une manière plus significative permettant ainsi leurs analyse et leurs visualisation contrairement au graphes d'origines qui ne peuvent même pas être charger en mémoire.

— Élimination du bruit : les grands graphes du web sont considérablement bruités, ils contiennent des liens et des nœuds erronés, ce bruit peut perturber l'analyse en faussant les résultats et en augmentant la charge de travail liée au traitement des données. la compression permet donc de filtrer le bruit et de ne mettre en évidence que les données importantes.

2.2.1 Motivations derrière la compression de graphes :

2.2.2 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 présentent : 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 2.1 – 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.

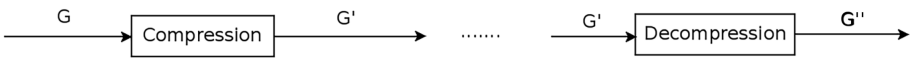

FIGURE 2.2 – Compression avec perte.

2.2.3 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 principaux 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 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.

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

Le CR est parfois appelé bit par bit (bpb) et il est définit alors comme étant le nombre moyen de bits requis pour stocker les données compressées (Uthayakumar et al., 2018). 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 liens (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 exprimer de deux manières différentes :

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

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

— **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.

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

2.2.4 Classification des méthodes de compression :

Le domaine de compression des graphes est un domaine qui a connu une grande évolution vu son importance. Une multitude de méthodes ont été proposées au cours des dernières années, ces méthodes diffèrent l'une de l'autre dans plusieurs points comme : le type de graphe en entrée, le type de structure en sortie, le type de compression et la technique utilisée pour la compression. En ce basant sur ces différences, plusieurs classifications ont été suggérées. Nous allons dans ce qui suit présenter les plus importantes parmi ces classifications.

Dans (Maneth and Peternek, 2015), les auteurs proposent une classification basée au début sur le type de compression, ils regroupent les méthodes en deux catégories principales : méthodes de compression sans perte et méthodes de compression avec perte. Les méthodes de compression sans perte sont divisées à leurs tour en trois classes selon le type de structures récupérées en sortie. La première classe est la représentation succincte, les méthodes de cette classe représentent le graphe sous forme d'une chaine de bits succincte irréversible lors de la décompression, la sortie de ces méthodes est une structure compacte du graphe originale. Parmi les méthode de cette classe, nous trouvons : Web Framework de Boldi et Vigna (Boldi and Vigna, 2004). La deuxième classe est la représentation structurelle. Contrairement à l'approche précédente, les méthodes de cette classe modifient la structure du graphe initial, sachant que les modifications apportées sont réversibles. La sortie sera donc une structure réduite et non pas compacte de la version initial. Entre ces méthodes, nous citons : RePair de Claude et Navarro (**?**). La dernière catégorie est la compression RDF qui est assez récente. Les méthodes de cette classe sont appliquées sur les bases de donnés RDF, nous trouvons parmi ces techniques : Dcomp de (Martínez-Prieto et al., 2012) . Les méthodes de compression avec perte quand à eux apportent des modifications irréversibles sur le graphe en supprimant les informations redondantes et le bruit. Comme exemple, nous citons : ASSG de (Zhang et al., 2014a).

Une autre classification a été exposée par Lui et al. dans (Liu et al., 2018a) qui classe les méthodes sur trois niveaux. Au premier et deuxième niveaux, les techniques de compression sont regroupées en fonction du type de graphe en entrée selon deux critères : graphe statique ou dynamique et graphe simple ou étiqueté. Pour le troisième niveau, les auteurs catégorisent les méthodes selon la technique de traitement utilisé. Quatre catégories sont définies : les méthodes de regroupement où agrégation, ces méthodes permettent d'agréger de manière récursive un ensemble de nœuds ou liens ou carrément un cluster en un super nœud ou un nœud virtuel, comme exemple de ces techniques, nous trouvons Grass (LeFevre and Terzi, 2010). Le deuxième type de méthodes sont les méthodes de compression de bits, ces méthodes minimisent le nombre de bits nécessaire au stockage du graphe en se basant sur le principe de description minimal MDL, elles peuvent être sans ou avec perte, parmi eux, nous citons LSH-based (Khan et al., 2014). La troisième classe est les méthodes de simplification qui suppriment les arrêtes inutiles et inintéressantes, entres ces méthodes, nous trouvons (Shen et al., 2006). La dernière catégorie est l'influence, les méthodes de cette catégorie décrivent le graphe par les flux d'influence les plus importantes ce qui permet de l'analyser plus facilement, ces méthodes permettent de formuler le problème de compression comme un processus d'optimisation dans lequel la quantité de données liée à l'influence est maintenue en sortie, parmi ces technique, nous motionnons (Shi et al., 2015).

La dernière classification que nous allons voir est la classification proposée dans le master de l'année passée par Belhocine et Guermah (W. GUERMAH, 2018). Cette classification s'appuie sur les taxonomies des différents techniques de compression existantes dans la littérature en se basant sur les classifications précédentes. Elle regroupe six classes de méthodes : basée sur l'ordre des nœuds en exploitant les principes de similarité et de localité du graphe, basée sur l'ordre des nœuds en exploitant la linéarisation du graphe, basée sur l'étiquetage des nœuds par intervalles, basée sur l'agrégation des nœuds, basée sur agrégation des liens.

Après l'étude des méthodes basées sur l'agrégation par extraction de motifs, nous avons constatés que certaines classes ne sont pas bien définit. Les imperfections que nous avons remarqués peuvent être énumérer dans ce qui suit : 1) les méthodes d'extraction de motifs englobent certaines méthodes d'agrégation et non pas le contraire, 2) Les méthodes d'extraction de motifs ne sont pas toutes des méthodes agrégatives, nous trouvons parmi eux d'autres méthodes basées sur un vocabulaire. Nous avons donc apportés des modifications sur cette classification. La figure ci-dessous représente la classification de l'année passé après raffinement où nous avons marqué nos apports en rouge.

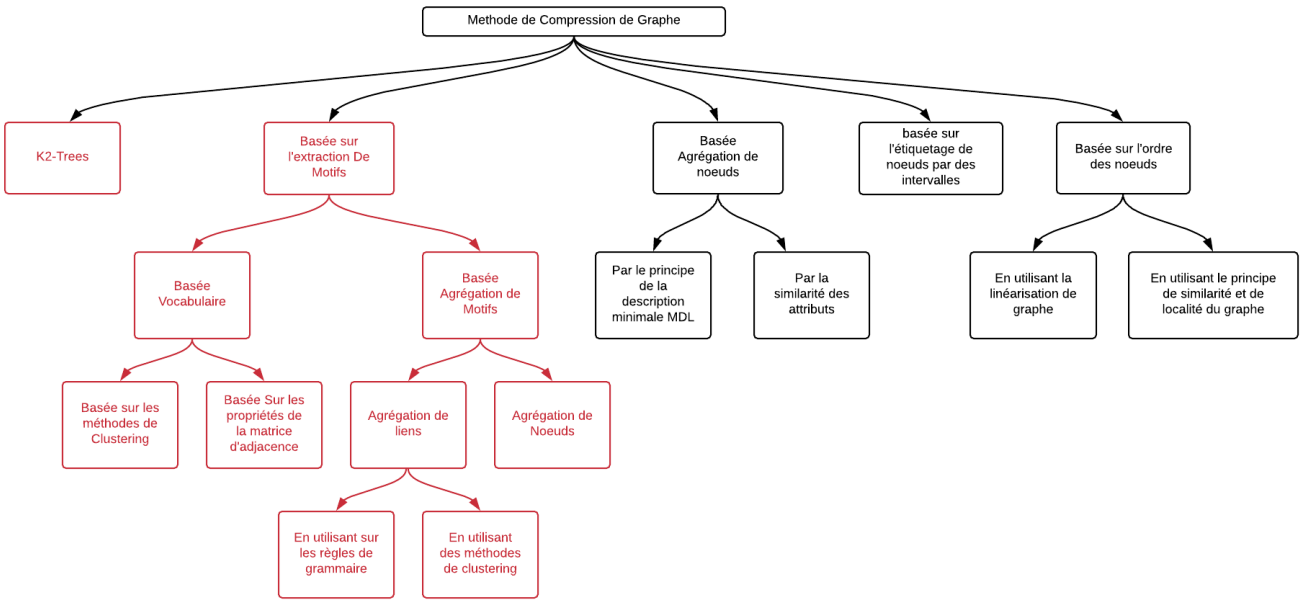

FIGURE 2.3 – Classification proposées des méthodes de compression

2.3 Compression par les K2-Trees

k^2-tree est une structure de données dense conçue à l'origine pour la compression des graphes du web, l'algorithme de base a été proposé par Bisaboa et al. dans leur article k^2-trees for Compact Web Graph. (Brisaboa et al., 2009). Elle a été appliquée ensuite dans d'autres travaux de compression comme les réseaux sociaux (Shi et al., 2012), les rasters (De Bernardo et al., 2013) et les bases de données RDF (Alvarez-Garcia et al., 2017).

En général, l'algorithme peut être appliqué à n'importe quelle matrice binaire. Dans le cadre de notre étude nous nous intéressons seulement à la matrice d'adjacence d'un graphe. k^2-trees exploite les propriétés de la matrice d'adjacence et tire parti des zones vides pour réduire l'espace de stockage et permettre au graphe de tenir en mémoire centrale. il offre aussi la possibilité de naviguer dans le graphe sans le décompresser, et de répondre aux requêtes de voisinage direct et inverse.

Étant donné une matrice d'adjacence A d'ordre n, k^2-trees représente A sous forme d'un arbre de recherche k^2-aires [1] de hauteur h = $[log_k$ n], chaque nœud contient un seul bit avec deux valeurs possibles : 1 pour les nœuds internes et 0 pour les feuilles, sauf le dernier niveau où les feuilles représentent les cases de A et peuvent prendre une valeur 0 ou 1. Chaque nœud interne de l'arbre a exactement k^2 fils. Avant la construction de l'arbre, il faut s'assurer que n est une puissance de k, dans le cas inverse, l'algorithme étend la matrice en rajoutant des zéros à droite et en bas de la matrice, l'ordre de la matrice deviens donc n'= $k^{log_k n}$.

1. Les arbres n-aires sont une généralisation des arbres binaires : chaque nœud a au plus n fils.

Pour construire l'arbre, k^2-trees commence par diviser la matrice en k^2 sous matrices d'ordre n/k, la racine correspond à la matrice complète, chaque sous matrice représente un nœud dans le premier niveau de l'arbre, elle est ajoutée comme un fils à la racine suivant un ordre de gauche à droite et de haut en bas. Le nœud est à 1 si la sous matrice qu'il représente contient au moins un 1, et à 0 si elle ne contient que des 0. Le processus est répété de manière récursive sur les sous matrices représentées par des 1. k^2 sous matrices sont considérées à chaque subdivision. L'opération est répétée jusqu'à ce que la subdivision atteint les cases de la matrice qui représenterons les feuilles de l'arbre au dernier niveau.

La figure 2.4 illustre la représentation k^2-trees d'une matrice de taille 10×10, étendue à une taille 16×16 pour un k=2 (Brisaboa et al., 2015).

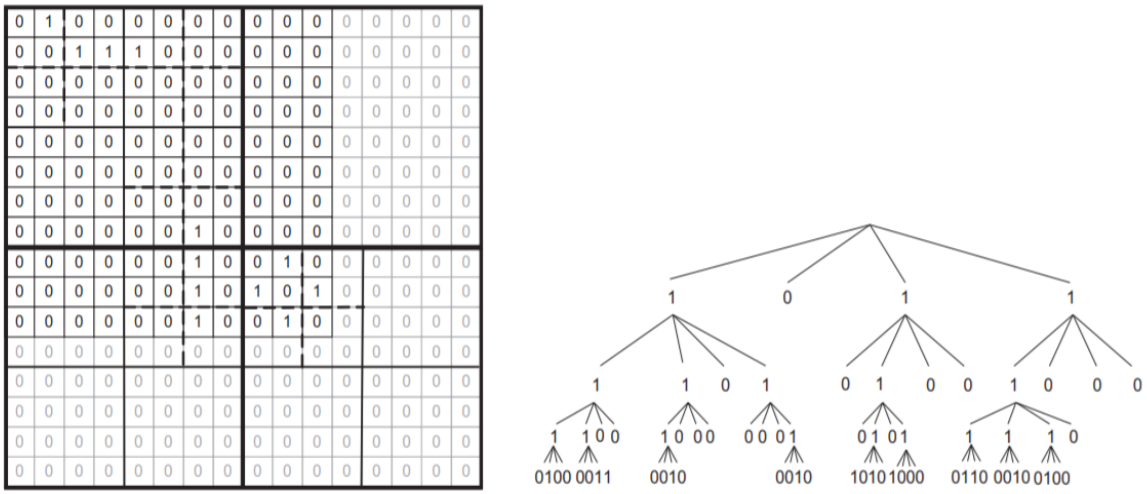

FIGURE 2.4 – Exemple de représentation k^2-trees

Pour le stockage de l'arbre, l'algorithme utilise deux tableaux de bits : un tableau T (Tree) contenant tous les nœuds de l'arbre à l'exception du dernier niveau et un tableau L (Leaves) contenant les feuilles du dernier niveau. Les nœuds et les feuilles sont ordonnés selon un parcours en largeur de l'arbre. Ci-dessous les deux tableaux T et L de l'exemple précédent (figure 2.4) :

$$\text{T} = 1011\ 1101\ 0100\ 1000\ 1100\ 1000\ 0001\ 0101\ 1110$$
$$\text{L} = 0100\ 0011\ 0010\ 0010\ 1010\ 1000\ 0110\ 0010\ 0100$$

Dans le pire des cas, l'espace totale pour la description de la structure est $k^2\text{m}(log_{k^2}\frac{n^2}{m} + O(1))$, où n est le nombre de nœuds du graphe et m le nombre de liens. Cependant, pour les graphe du monde réel, l'espace nécessaire pour le stockage est bien meilleur.

Dans le même travail (Brisaboa et al., 2009), et dans le but d'obtenir un compromis entre la taille de l'arbre et le temps de parcours, les auteurs proposent une hybridation qui

consiste à changer la valeur du paramètre k en fonction du niveau de l'arbre en donnant à k une grande valeur au début pour réduire le nombre de niveaux et améliorer ainsi le temps de recherche, et une petite valeur à la fin pour avoir des petites sous matrices et réduire l'espace de stockage. Pour le stockage de l'arbre, un tableau T_i est utilisé pour chaque valeur k_i, le tableau L reste le même.

Plusieurs variantes de l'algorithme de base ont été proposées dans la littérature dont le but était soit d'obtenir un meilleur résultat de compression, soit d'appliquer la méthode sur d'autres types de graphes. Nous allons dans ce qui suit présenter les principaux travaux qui traitent cela.

Dans (Shi et al., 2012), les auteurs proposent deux techniques d'optimisation de l'algorithme : la première consiste à trouver un certain ordre des nœuds qui permet de regrouper les 1 de la matrice d'adjacence dans une seule sous matrice au lieu qu'ils soient dispersés de manière aléatoire. La recherche d'un ordre optimal des nœuds est inenvisageable, avec k=2, le problème peut être réduit à un autre problème (min bisection) qui est NP-difficile, le problème se complique davantage lorsque k est dynamique. Les auteurs utilisent alors DFS avec des heuristiques pour trouver une approximation de l'ordre optimal. Cette optimisation permet de réduire le nombre de nœuds internes et produit ainsi un arbre optimale. La deuxième optimisation est de trouver la valeur de k la plus adéquate pour chaque nœud interne, calculer cette valeur pour chaque nœud peut engendrer un temps de calcul très important. Pour éviter cela, les auteurs affectent la même valeur k pour les nœuds ayant le même parent.

Dans un travail ultérieure (Brisaboa et al., 2014b), les auteurs de l'article de base apportent deux améliorations principales de leur méthode dans le but d'optimiser l'espace et le temps de parcours de l'arbre produit : la première est de construire k^2 arbres distincts pour les k_0^2 sous matrices du premier niveau. Elle présente plusieurs avantages : (1) l'espace est réduit étant donné que la taille de chaque arbre est en fonction de $\frac{n^2}{k^2}$, (2) le temps de parcours s'améliore puisque T et L sont plus petits. La deuxième amélioration est la compression de L qui consiste à construire un vocabulaire V de tous les sous matrices du dernier niveau sous forme de séquences de bits, les classer par fréquence d'apparition et remplacer leurs occurrences dans L par des pointeurs, cela permet d'éviter la redondance et réduire par suit la taille de la structure. Les pointeurs sont représentés par des codes de longueur variable ordonnés, le plus petit correspond a la sous matrice la plus fréquente. Néanmoins, cette représentation ne permet pas un accès direct dans L étant donné qu'une décompression séquentiel est nécessaire pour récupérer une position. Pour remédier à ce problème, les auteurs utilisent le principe de Directly Addressable Codes (DACs) [2] (Brisaboa et al., 2013) pour garantir un accès rapide au pointeur et conserver ainsi une navigation efficace.
Exemple : Pour la figure 2.4, le vocabulaire et L sont représentés comme suit :

2. DAC est une technique qui encode une séquence d'entiers ou mots en utilisant une structure à longueur variable, son avantage principale est l'accès direct au mot sans passer par le décodage.

$$V = [0010\ 0100\ 0011\ 1010\ 1000\ 0110]$$
$$\mathrm{L} = c_1 c_2 c_0 c_0 c_3 c_4 c_5 c_0 c_1$$

Dans (Brisaboa et al., 2012), les même auteurs développent la représentation k^2-trees pour les graphes dynamiques. Ils proposent une nouvelle structure nommée dk^2-trees pour *Dynamique k^2-trees* qui offre les même capacités de compression et fonctionnalités de navigation que le cas statique et qui permet également d'avoir des mises à jour sur le graphe. Pour atteindre ces objectifs, dk^2-trees remplace la structure statique de k^2-trees par une implémentation dynamique. Dans cette nouvelle implémentation, les deux tableaux T et L sont remplacés par deux arbre, nommés T_{tree} et L_{tree} respectivement. Les feuilles de T_{tree} et L_{tree} stockent des parties des bitmaps T et L. La taille des ces feuilles est une valeur paramétrée. Les nœuds internes des deux arbres permettent d'accéder aux feuilles et de les modifier. Chaque nœud interne de T_{tree} contient trois(03) éléments : deux compteurs b et o qui contiennent respectivement le nombre de bits et le nombre de "1" stockés dans les feuilles descendantes de ce nœud, un pointeur P vers le nœud fils. Les nœuds internes de L_{trees} sont similaires sauf qu'ils ne contiennent que b et P. Avec cette structuration, T_{tree} et L_{tree} permettent l'ajout et la suppression des liens dans le graphe. La figure 2.5 présente une représentation dk^2-trees (Brisaboa et al., 2012) :

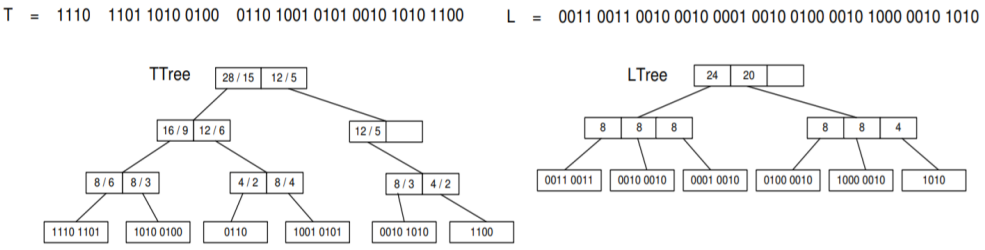

FIGURE 2.5 – Exemple d'une représentation dk^2-trees

Sandra and al. (De Bernardo et al., 2013) présentent k^n-trees, une généralisation des k^2-trees pour les problèmes multidimensionnelles. Cette méthode possède plusieurs applications, elle est utilisée pour représenter les bases de données multidimensionnelles, les rasters et les graphes dynamiques. k^n-trees repose sur k^2-trees pour représenter une matrice à n-dimensions, la matrice est décomposer en k^n sous-matrices de même taille, comme suit : Sur chaque dimension, K-1 hyperplans devise la matrice dans les position i*$\frac{n}{K}$,pour $i \in [1, K-1]$. Une fois les dimensions partitionnés, k^n sous-matrices sont induites, elles sont représentées par des nœuds dans l'arbre comme dans l'algorithme de base. Les structures utilisées pour le stockage sont aussi les même (T et L). En posant n=3, la méthode peut être appliquée sur les graphes dynamiques ou temporelles. Ce type de graphe est représenté par une grille à 3 dimensions $X \times Y \times T$, où les deux premières dimensions représentent les nœuds de départ et de destination, et la troisième dimension représente le temps. La figure 2.6 présente une représentation k^3-trees d'un graphe dynamique (de Bernardo Roca, 2014)

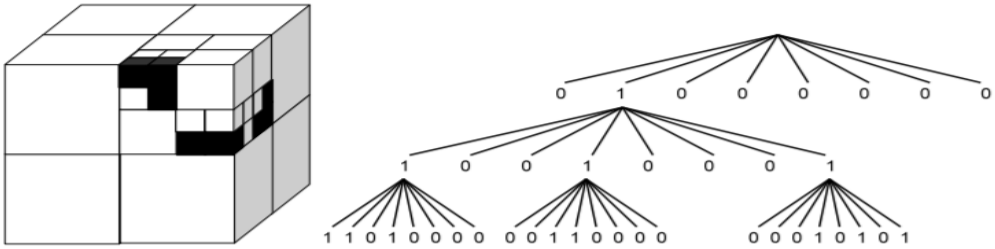

FIGURE 2.6 – Exemple d'une représentation k^3-trees

Une autre variante a été proposée par (de Bernardo Roca, 2014). La représentation de base regroupe seulement les zones de zéros, puisque elle a été conçue au début pour les graphes du web qui possèdent une matrice d'adjacence extrêmement creuse. Les auteurs proposent d'étendre cette représentation en regroupant les zones de "1" également. L'idée générale est d'arrêter la décomposition de la matrice d'adjacence quand une zone unis est trouvé, à savoir des zéros ou des uns. Pour distinguer entre les différents nœuds, une représentation quadtree est utilisée (De Berg et al., 1997) : une couleur est attribuée à chaque nœud, blanc pour une zone de zéro, noir pour une zone de uns et gris pour les nœuds internes i.e les zones contenant des uns et des zéros. Pour le stockage des nœuds, les auteurs proposent quatre encodages présentés par la suite :

k^2-$trees1^{2-bits-naive}$ **:** Dans cet encodage, deux bits sont utilisés pour représenter chaque type de nœud. L'attribution des bits n'est pas arbitraire, le premier bit du poids fort indique si le nœud est un nœud interne (0) ou une feuille (1), le deuxième détermine si les feuilles sont blanches (0) ou noires (1). Nous aurons donc : "10" pour les nœuds gris, "01" pour les nœuds noirs et "00" pour les nœuds blancs. Notant que les feuilles du dernier niveau sont représentés par un seul bit. Après le codage, le premier bit de chaque nœud sauf ceux du dernier niveau est stocké dans T, un autre tableau T' est créé pour sauvegarder le deuxième bit, les nœuds du dernier niveau sont stockés dans L.

k^2-$trees1^{2-bits}$ **:** Le même principe de l'encodage précédant sauf que les nœuds gris sont représentés par un seul bit, toujours à "1", le tableau T' va contenir dans ce cas la couleur des feuilles ce qui va réduire la taille de la structure.

k^2-$tree1s^{DF}$ **:** Cet encodage est similaire à k^2-$trees^{2-bits}$, mais il utilise un seul bit pour les nœuds blancs et deux bits pour les nœuds noirs et gris, compte tenue de la fréquence des nœuds blancs dans les graphes du monde réel par rapport au autres. Nous aurons donc : "0" pour les nœuds blancs, "10" pour les nœuds gris et "11" pour les nœuds noirs.

k^2-$trees1^{5-bits}$ **:** Le dernier encodage repose sur la représentation de base. Un nœud blanc est représenté par "0", un nœud noir ou gris par "1", exactement comme le k^2-trees d'origine. Pour identifier un nœud noir (zone de uns), il sera représenté par une combinaison impossible : k^2 fils de "0" sont ajoutés aux nœuds noirs pour les distinguer.

La figure 2.7 illustre une représentation k^2-tress1 avec les quatre encodages (de Bernardo Roca, 2014) :

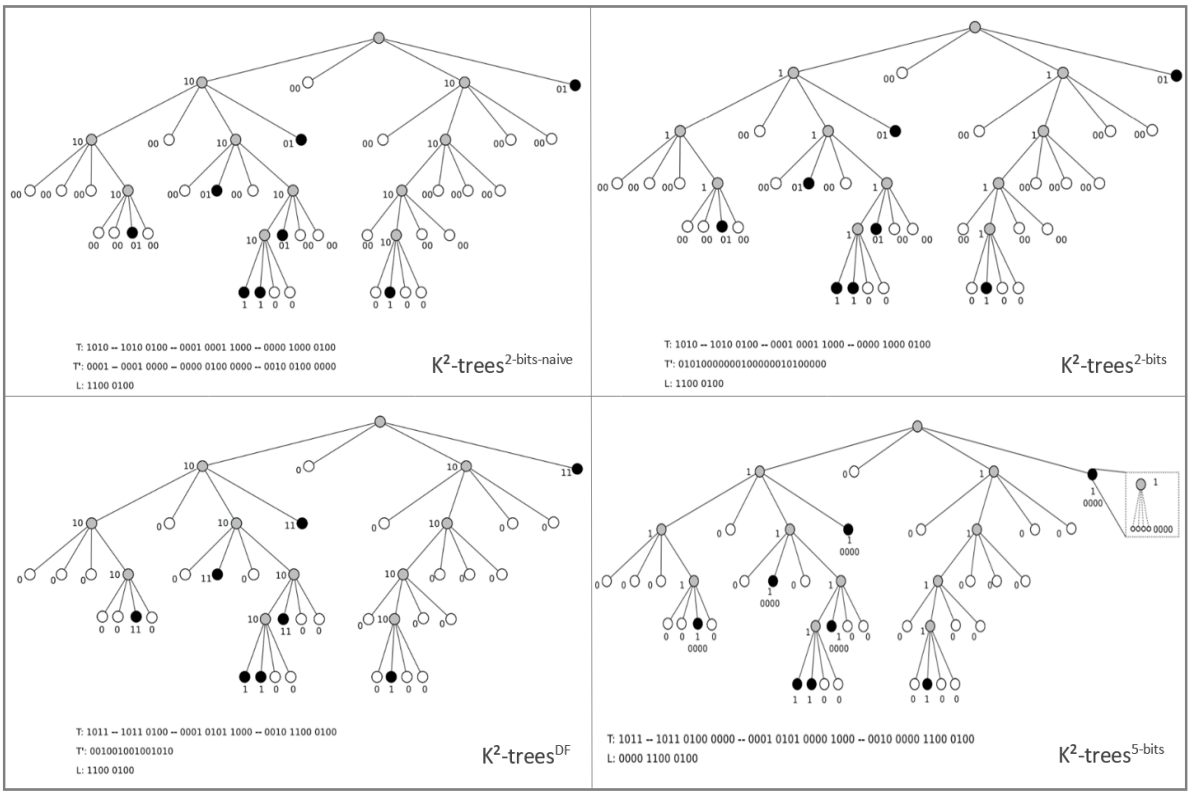

FIGURE 2.7 – Exemple d'une représentation k^2-trees1 avec les quatre encodages

Dans (Zhang et al., 2014b), les auteurs proposent Delta-k^2-trees, une variante qui exploite la propriété de similarité entre les nœuds voisins du graphe pour réduire le nombre de uns dans la matrice d'adjacence. Notons par *Matrix* la matrice d'adjacence, Delta-k^2-trees construit une nouvelle matrice appelée *Delta-matrix*, une ligne i de *Delta-matrix* va contenir la différence entre les deux lignes *Matrix*[i] et *Matrix*[i-1] si cela décroit le nombre de uns sinon elle sera égale à *Matrix*[i] i.e :

$$\begin{cases} Si\ count1s(matrix[i]) < countDif(matrice[i], matrix[i-1]) \\ \quad Delta - matrix[i] := matrix[i] \\ Sinon \\ \quad Delta - matrix[i] := matrix[i] \bigoplus matrix[i-1] \end{cases}$$

Où *count1s* compte le nombre de 1 dans une ligne, *countDif* compte le nombre de bits différents entre deux lignes et $\bigoplus$ représente le "ou exclusif".

Pour déterminer si une ligne est identique à celle de la matrice d'adjacence ou non, un tableau D est utilisé : Si D[i]=1 la ligne est identique, sinon c'est une ligne de différence. La matrice *Delta-matrix* contient moins de uns que la matrice d'adjacence, d'où elle est plus creuse, ce qui permet de réduire la taille de la structure et avoir un meilleur taux de compression. Cependant le temps de parcours est plus grand car pour accéder a certaines lignes (lignes de différence), le graphe doit être décompresser et la matrice d'adjacence reconstituer.
La figure 2.8 présente un exemple de la représentation Delta-k^2-trees (Zhang et al., 2014b).

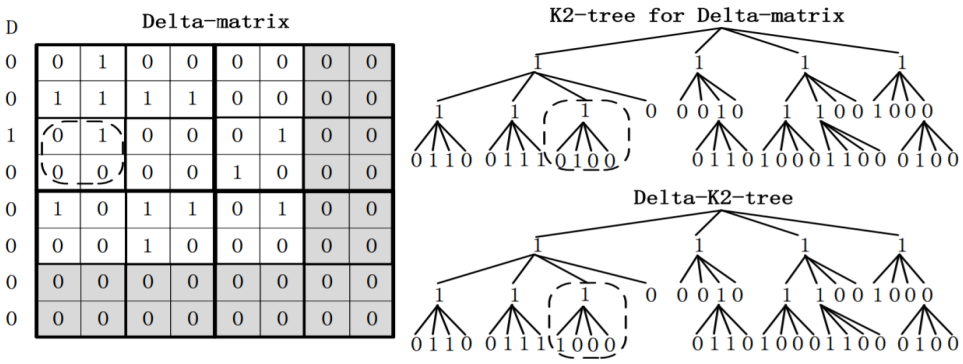

FIGURE 2.8 – Exemple d'une représentation Delta-k^2-trees

K^2-treaps est une autre variante de k^2-trees,elle a été proposée dans (Brisaboa et al., 2014a). Cette variante combine les k^2-trees avec une autre structure de données appelée treaps [3] (Aragon and Seidel, 1989). Les auteurs appliquent cette méthode sur des grilles multidimensionnelles comme OLAP pour pouvoir les stocker et répondre efficacement aux requêtes top-K (Badr, 2013). La méthode peut être également appliquée sur les graphes pondérés, où chaque case de la matrice d'adjacence du graphe comporte le poids de l'arête qu'elle représente au lieu d'un 1. Comme dans l'algorithme de base, une décomposition récursive en k^2 sous-matrices est appliquée sur la matrice d'adjacence et un arbre k^2-air est construit , comme suit : la racine de l'arbre va contenir les coordonnées ainsi que la valeur de la cellule ayant le plus grand poids de la matrice. La cellule ajoutée à l'arbre est ensuite supprimée de la matrice. Si plusieurs cellules ont la même valeur maximale, l'une d'elles est choisis au hasard. Ce processus est répété récursivement sur chaque sous matrice en choisissant la cellule la plus lourde qu'elle contient pour la représenter dans l'arbre et la supprimer de la sous-matrice. La procédure continue sur chaque branche de l'arbre jusqu'à ce qu'on tombe sur les cellules de la matrice d'origine ou sur une sous matrice complètement vide (contient que des zéros).
La figure 2.9 suivante illustre la représentation k^2-treaps d'un graphe pondéré (Badr, 2013) :

3. Les Treaps sont des arbres de recherche binaire avec des nœuds ayant deux attributs : clés et priorité, la recherche dans ces arbres s'effectue selon ces attributs.

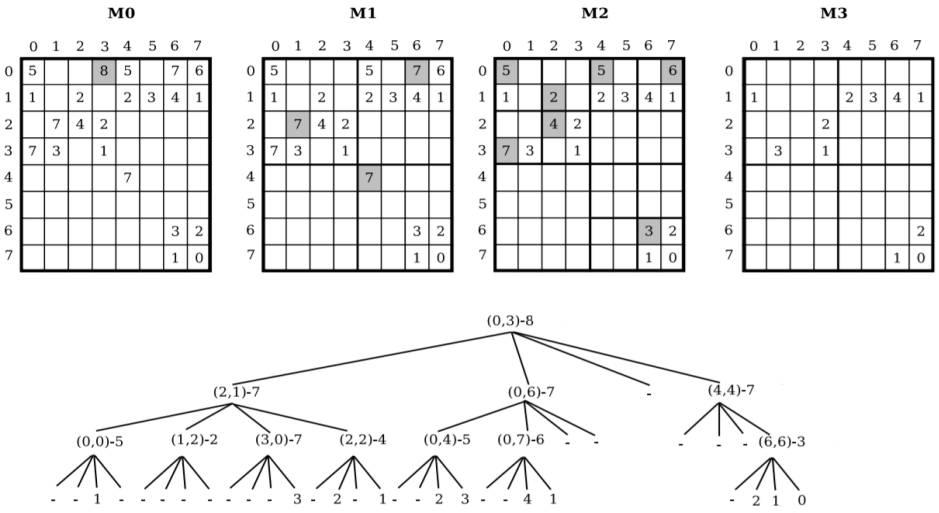

FIGURE 2.9 – Exemple d'une représentation k^2-treaps d'un graphe pondéré

Pour avoir une bonne compression, k^2-treaps effectue des transformations sur les données stockées. La première transformation consiste à changer les coordonnées représentées dans l'arbre en des coordonnées relatives par rapport à la sous matrice actuelle. La deuxième est de remplacer chaque poids dans l'arbre par la différence entre sa valeur et celle de son parent.
Trois structures de données sont utilisées pour sauvegarder les coordonnées et les valeurs des cellules ainsi que la topologie de l'arbre. Chaque structure est détaillée dans ce qui suit :

— *Listes de coordonnées locales :* La séquence de coordonnées de chaque niveau l de l'arbre est stockée dans une liste $coord[l]$.

— *Liste des valeurs :* Le parcours de l'arbre se fait en largeur, la séquence des valeurs récupérées est stockée dans une liste nommée *values*. Un tableau nommé *first* est utilisé pour sauvegarder la position du commencement de chaque niveau dans *values*.

— *L'arbre :* La structure de l'arbre k^2-treaps est sauvegardée avec un arbre k^2-trees, les nœuds contenant des valeurs dans k^2-treaps sont représentés par des uns, les nœuds vides par des zéros. Pour le stockage de l'arbre, un seul tableau T est utilisé.

La figure 2.10 représente les structures de données utilisées pour le stockage de l'arbre de la figure 2.9 précédente (Badr, 2013) :

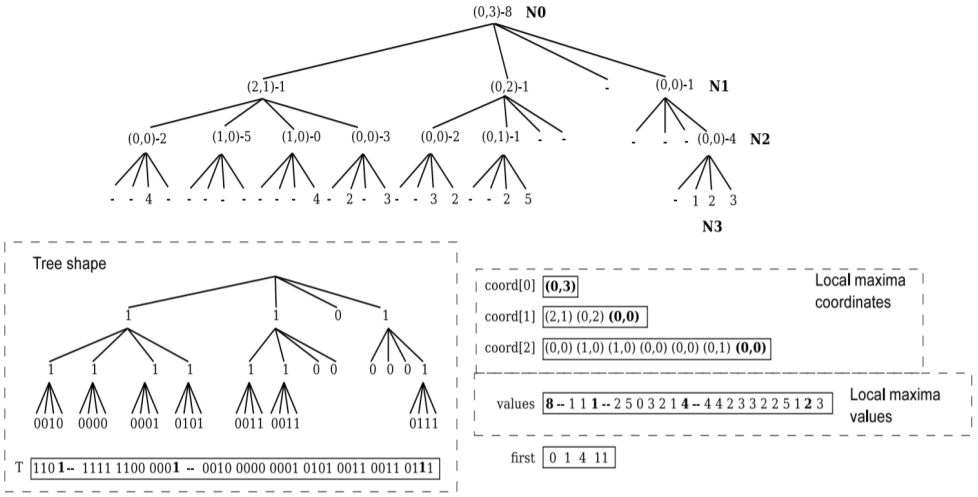

FIGURE 2.10 – Structures de données pour une représentation k^2-treaps

Garcia et al. (Garcia et al., 2014) proposent Ik^2-trees pour Interleaved k^2-trees. Elle est appliquée sur les bases de données RDF ainsi que sur les graphes dynamiques. Dans les graphes dynamiques, les deux premières dimensions correspondent aux nœuds source et destination et la troisième dimension reflète le temps. Le graphe est donc définit par |T| matrices d'adjacence prisent à des instants t_k différents. Ik^2-trees représente les matrices simultanément, chaque matrice est représentée par un arbre k^2-trees, les arbre sont par la suite regroupés dans un seul arbre. Chaque nœud de l'arbre obtenu représente une sous-matrice comme dans l'algorithme de base, sauf qu'au lieu d'utiliser un seule bit, Ik^2-trees utilise 1 à |T| bits pour représenter le nœud. Le nœud racine contient |T| bits. Le nombre de bits de chaque fils dépend du nombre de uns de son parent. L'arbre finale est stocké comme dans l'original k^2-trees avec deux tableau : T et L. La figure 2.11 est un exemple de Ik^2-trees appliqué sur un graphe dynamique représenté sur trois instants (Garcia et al., 2014) :

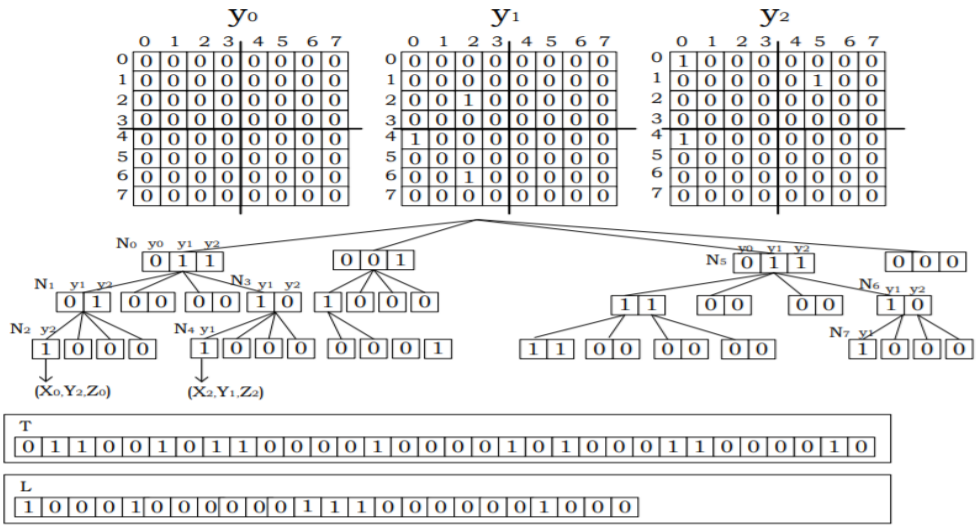

FIGURE 2.11 – Exemple d'une représentation Ik^2-trees

Une variante de Ik^2-trees appelée Differential Ik^2-tree a été étudiée dans (Alvarez-Garcia et al., 2017). Elle vise à améliorer le taux de compression en représentant uniquement les changements survenus sur le graphe à un instant t_i au lieu d'une instance complète : A l'instant t_0, une capture complète du graphe (matrice d'adjacence) est stockée. A l 'instant t_k, pour k>0, seules les arrêtes qui changent de valeurs entre t_{k-1} et t_k sont stockées. Les matrices sont représentées à la fin de la même manière que Ik^2-trees. La limite de cette représentation est que la structure doit être décompresser lors d'une requête. La figure 2.12 montre un exemple d'une représentation diff Ik^2-trees (Alvarez-Garcia et al., 2017).

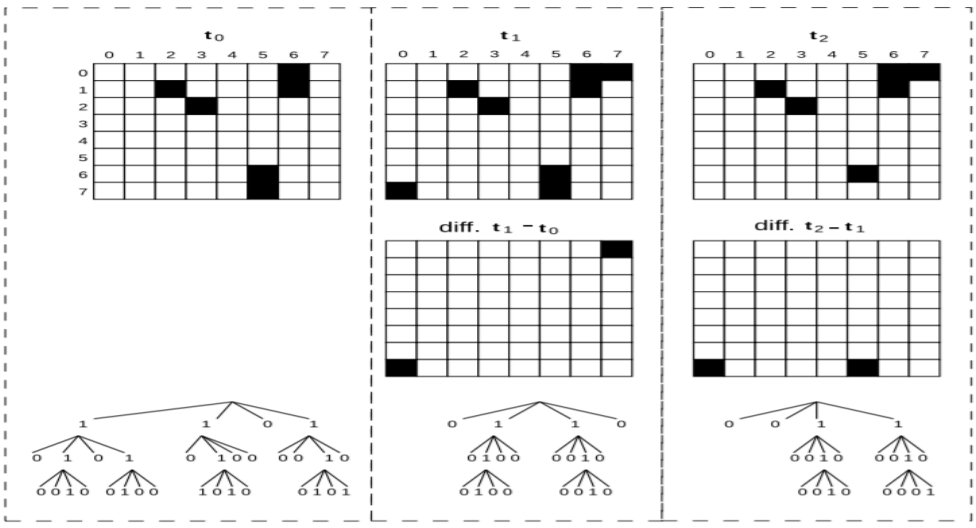

FIGURE 2.12 – Exemple d'une représentation diff Ik^2-trees

Dans (Álvarez-García et al., 2018), les auteurs étendent la représentation k^2-trees pour les bases de données orientées graphes. Ces graphes sont étiquetés, attribués, orientés et

ont des arêtes multiples. Ils présentent le graphe sous forme d'une nouvelle structure intitulée Att k^2-trees pour Attributed k^2-trees. La figure 2.13 montre un exemple de graphe pris en compte par Att K^2-trees (Álvarez-García et al., 2018).

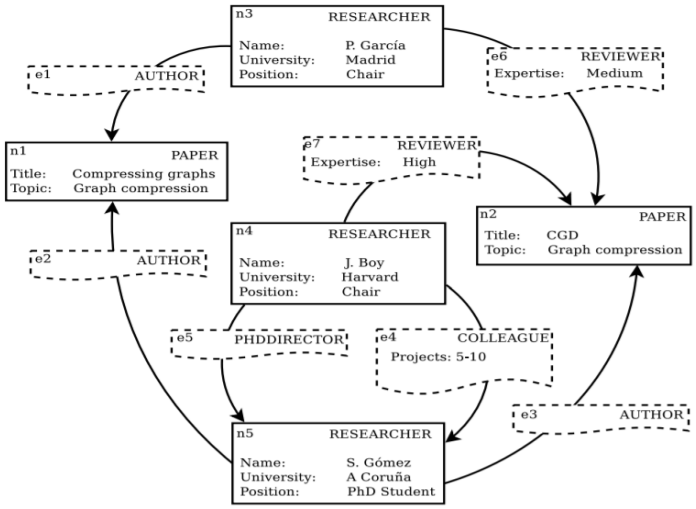

FIGURE 2.13 – Exemple d'un graphe étiqueté, attribué, orienté et multiple

Structures de donnés : La représentation obtenue par la compression est composée d'un ensemble d'arbres k^2-trees et d'autres structures supplémentaires. Le graphe est représenté par trois composants : un schéma de données, les données incluses dans les nœuds et les liens et finalement la relation entre les éléments du graphe. Chaque composant est présenté dans ce qui suit :

— *Schéma :* Ce composant gère les étiquettes et les attribues de chaque type d'éléments, il joue le rôle d'un indexe dans la représentation. Il est composé de :

Un schéma de nœuds : représenté par un tableau qui contient les étiquettes des nœuds ordonnées lexicographiquement. Un identifiant est attribué à chaque nœud du graphe selon l'ordre du tableau, les m_1 possédant la première étiquette du tableau vont avoir des identifiant de 1 à m_1, les m_2 nœuds avec la deuxième étiquette du tableau vont avoir des identifiants de m_1+1 à m_1+m_2 et ainsi de suit. chaque entrée du tableau va stocker le plus grand identifiant portant son étiquette, cela permet de trouver l'étiquette d'un nœud à travers son identifiant.

Un schéma d'arêtes : Comme dans le cas des nœuds, un tableau est utilisé pour stocker les étiquettes des arêtes avec le même principe.

Le schéma est le point de départ de la représentation, il permet d'obtenir l'étiquette d'un nœud ou d'une arête, et d'accéder à ses attributs.

— *Données :* Ce composant contient tous les valeurs que peut prendre un attribut dans le graphe. Un attribut peut être représenté de deux façons différentes selon sa fréquence d'apparition, on distingue donc deux type d'attributs :

Attributs rares : Ce sont les attributs qui prennent généralement des valeurs différentes à chaque apparition, ils sont stockés dans des listes et indexés avec l'identifiant de l'élément.

Attributs fréquents : Ce type d'attributs est sauvegardés dans deux matrices, une pour les attributs des nœuds et l'autre pour les attributs des liens. Les matrices sont stockées sous forme d'arbres k^2-trees.

La figure 2.14 illustre les deux composants schéma et données de la représentation Attk^2-trees de la figure 2.13 précédente (Álvarez-García et al., 2018) :

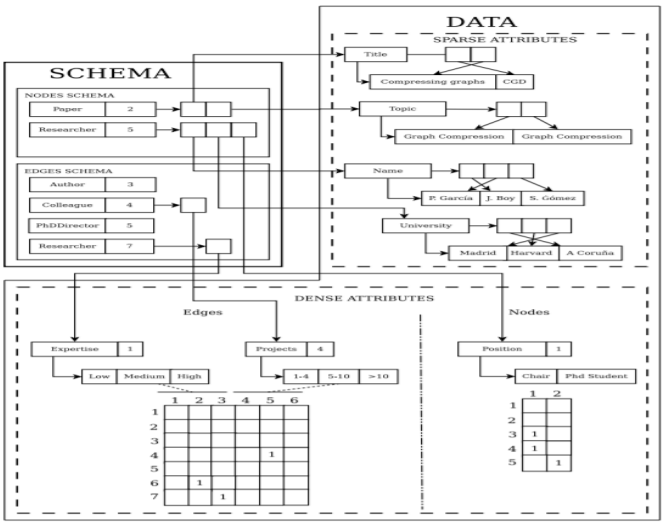

FIGURE 2.14 – Exemple d'une la représentation Attk^2-trees (1/2)

— *Relations :* C'est le dernier composant de Attk^2-trees, il stocke les relations entre les nœuds et les arêtes du graphe en utilisant un arbre k^2-trees et d'autres structures pour sauvegarder les identifiants des arêtes ainsi que les arêtes multiples. Les structures supplémentaires sont les suivantes :

Multi : Un tableau qui indique si l'arête est multiple ou non.

Firt : Un Tableau qui donne l'identifiant de l'arête, ou de celui de la première dans le cas d'une arrête multiple.

Next : Un tableau qui contient les identifiants des arêtes multiples restantes.

La figure 2.15 donne la représentation Attk^2-trees des relations du graphe de la figure 2.13 (Álvarez-García et al., 2018) :

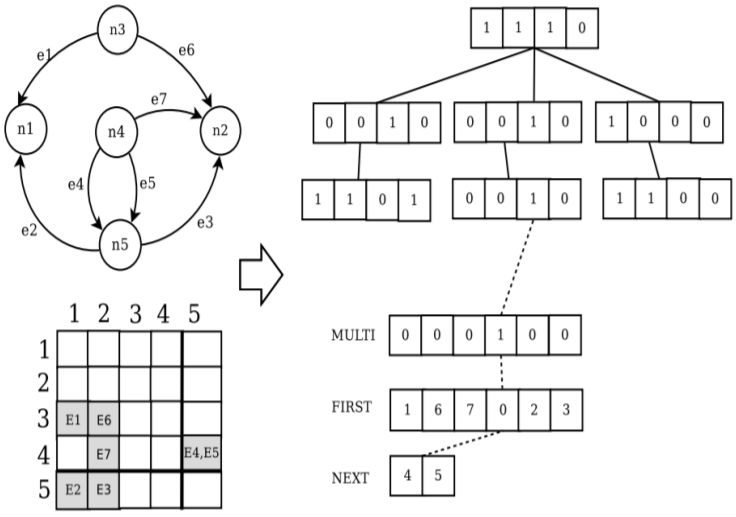

FIGURE 2.15 – Exemple d'une la représentation Attk^2-trees (2/2)

Dans le même article (Álvarez-García et al., 2018), les auteurs étendent Attk^2-trees pour les graphes dynamiques, ils proposent une nouvelle variante appelée dynAttk^2-trees qui supporte le changement dans les attributs et les liens du graphe. Comme Attk^2-trees, dynAttk^2-trees représente le graphe avec trois composants : Schémas, données et relations. Les composants sont semblables à ceux de Attk^2-trees mais avec certaines amélioration vu la nature dynamique du graphe.

Structure de données :

— *Schéma :* En ce qui concerne les nœuds, leurs étiquettes sont stockées dans une liste dynamique ordonnée lexicographiquement. En outre, une séquence dynamique est utilisée pour sauvegarder le type de chaque nœud. Elle est stockée ensuite sous forme d'un arbre d'ondelettes [4] (Grossi et al., 2003). Le même principe est appliqué sur les arrêtes.

— *Données :* Les attributs rares sont stockés dans des listes dynamiques, quant aux attributs fréquents, ils sont sauvegardés avec des arbres dk^2-trees (un arbre pour chaque attribut).

— *Relations :* Le stockage des relations se fait à l'aide d'un dk^2-trees et des tableaux dynamiques pour stocker les identifiants des arêtes et les arêtes multiples.

4. Ou wavelet en anglais, est un arbre binaire équilibré qui contient des données compressées dans une représentation optimale.

Article	Graphe en entrée				Compression		Structure en sortie		Graphe de test	Résultat
	Orienté	Non orienté	Statique	Dynamique	Autre Propriétés	Avec perte	Sans perte	Succincte	Structurelle		
k^2-trees : Algorithme de base (Brisaboa et al., 2009)	✓	✓	✓	✗		✗	✓	✓	✗	eu-2005	5.21 bits/lien taux de compression : 16.88%
										cond-mat	taux de compression : 15.58%
k^2-trees : Hybridation (Brisaboa et al., 2009)	✓	✓	✓	✗		✗	✓	✓	✗	eu-2005	5.21 bits/lien
k^2-trees : Optimisation (Shi et al., 2012)	✓	✓	✓	✗		✗	✓	✓	✗	cond-mat	taux de compression : 37.96%
k^2-trees : Amélioration (Brisaboa et al., 2014b)	✓	✓	✓	✗		✗	✓	✓	✗	eu-2005	3.22 bits/lien
dk^2-trees (Brisaboa et al., 2012)	✓	✓	✗	✓		✗	✓	✓	✗	eu-2005	6.2 bits/lien
k^n-trees (De Bernardo et al., 2013)	✓	✓	✗	✓		✗	✓	✓	✗	CommNet	taux de compression : 65.16%

TABLE 2.1 – Synthèse des méthodes de compression par k^2-trees.

Article	Graphe en entrée					Compression		Structure en sortie		Graphe de test	Résultat
	Orienté	Non orienté	Statique	Dynamique	Autre Propriétés	Avec perte	Sans perte	Succincte	Structurelle		
k^2-trees1(de Bernardo Roca, 2014)	✓	✓	✓	✗		✗	✓	✓	✗	eu-2005	taux de compression : 16.23%
Delta-k^2-trees (Zhang et al., 2014b)	✓	✓	✓	✗		✗	✓	✓	✗	eu-2005	3.24 bits/lien
k^2-treaps (Brisaboa et al., 2014a)	✓	✓	✓	✗	Pondéré	✗	✓	✓	✗	SalesDay	2.48 bits/lien
Ik^2-trees (Garcia et al., 2014)	✓	✓	✗	✓		✗	✓	✓	✗	CommNet	taux de compression : 60.43%
Diff Ik^2-trees (Alvarez-Garcia et al., 2017)	✓	✓	✗	✓		✗	✓	✓	✗	CommNet	taux de compression : 60.43%
Att k^2-trees (Álvarez-García et al., 2018)	✓	✓	✓	✗	Étiqueté Attribué Multiple	✗	✓	✓	✗	Movielen-10M	taux de compression : 89.97%
dynAtt k^2-trees (Álvarez-García et al., 2018)	✓	✓	✗	✓	Étiqueté Attribué Multiple	✗	✓	✓	✗	Movielen-10M	taux de compression : 93.75%

TABLE 2.2 – Synthèse des méthodes de compression par k^2-trees.

2.4 Compression par extraction de motifs

Les motifs fréquents sont des connaissances extraites sur des données. Leurs but est de fournir à l'utilisateur des informations non triviales, implicites, présumées non connues. Ils lui offrent ainsi une meilleure appréhension des données. Dès lors, l'extraction de motifs fréquents est devenue une tâche importante dans la fouille de données et un thème très étudié par la communauté. Elle a aussi été vastementutilisée dans le domaine de compression des graphes vu qu'elle permet de ne garder que l'information utile et d'éliminer les redondances de manière efficace. En effet, nous trouvons plusieurs méthodes basées sur ce principe où nous pourrons clairement distinguer deux grandes classes : (i) les méthodes de compression basées vocabulaire (ii) les méthodes de compression basées Agrégation.

Dans cette section, nous allons expliquer le principe de base de chaque classe où nous allons subdiviser chacune en plusieurs sous-classes en se basant sur ce dernier.

2.4.1 Compression basée vocabulaire

Les méthodes de compression par extraction de motifs basées vocabulaire sont des méthodes qui ont attirées l'attention des chercheurs ces dernières années car elles permettent une meilleur compréhension du graphe. Elles partent toujours d'un ensemble de structures prédéfinies qui ont été prouvées fréquentes dans les graphes réels. Deux sous classes de cette dernières peuvent être identifier :

Basées sur des méthodes de clustering

Les méthodes de cette classe s'appuient sur le fait qu'on ne peut pas comprendre facilement les graphes denses, alors que quelques structures simples sont beaucoup plus faciles à comprendre et souvent très utiles pour analyser le graphe. Elles se basent sur des algorithmes de détection de communauté. La question suivante peut alors se poser : pourquoi ne pas appliquer l'un des nombreux algorithmes de détection de communauté ou de partitionnement de graphe pour compresser le graphe en termes de communautés ? La réponse est que ces algorithmes ne servent pas tout à fait le même objectif que la compression. Généralement, ils détectent de nombreuses communautés sans ordre explicite, de sorte qu'une procédure de sélection des sous-graphes les plus «importants» est toujours nécessaire. En plus de cela, ces méthodes renvoient simplement les communautés découvertes, sans les caractériser (par exemple, clique, étoile) et ne permettent donc pas à l'utilisateur de mieux comprendre les propriétés du graphe.

Parmi les méthodes de détection les plus importantes dans la littérature nous trouvons :

1. **METIS(Karypis and Kumar, 2000)** :est un schéma de partitionnement de graphe multiniveau basé sur la bissection récursive multiniveau (MLRB). Tant que la taille du graphe n'est pas sensiblement réduite, il grossit d'abord le graphe d'entrée en regroupant les nœuds dans des super-nœuds de manière itérative, de sorte que la coupe des bords soit préservée. Ensuite, le graphe grossi est partitionné à l'aide de MLRB et le partitionnement est projeté sur le graphe d'entrée G par le biais du retour en arrière. La méthode produit k partitions à peu près égales.

2. **SPECTRAL(Hespanha, 2004)** : partitionne un graphe en effectuant une classification en k-means sur les k premiers vecteurs propres du graphe d'entrée. L'idée derrière cette classification est que les nœuds avec une connectivité similaire ont des scores propres similaires dans les k premiers vecteurs.
3. **LOUVAIN(Blondel et al., 2008)** :est une méthode de partitionnement basée sur la modularité pour détecter la structure de communauté hiérarchique. LOUVAIN est une méthode itérative : (i) Chaque nœud est placé dans sa propre communauté. Ensuite, les voisins j de chaque nœud i sont pris en compte et i est déplacé vers la communauté j si le déplacement produit le gain de modularité maximum. Le processus est appliqué à plusieurs reprises jusqu'à ce qu'aucun gain supplémentaire ne soit possible. (ii) Un nouveau graphe est créé dont les super nœuds représentent les communautés et les super arêtes sont pondérés par la somme des poids des liens entre les deux communautés. L'algorithme converge généralement en quelques itérations.
4. **SLASHBURN (Kang and Faloutsos, 2011)** :est un algorithme de ré-ordonnancement de nœud initialement développé pour la compression de graphes. Il effectue deux étapes de manière itérative : (i) il supprime les nœuds de haute centralité du graphe (ii) Il réorganise les nœuds de manière à ce que les identificateurs les plus petits soient attribués aux nœuds de degré élevé et les nœuds des composants déconnectés obtiennent les identificateurs les plus grands. Le processus est répété sur le composant connecté.
5. **BIGCLAM(Yang and Leskovec, 2013)** :est une méthode de détection de communauté à chevauchement évolutive. Elle est construite sur le constat que les chevauchements entre les communautés sont étroitement liés. En modélisant explicitement la force d'affiliation de chaque couple nœud-communauté, un facteur latent non négatif est attribué à cette dernière, qui représente le degré d'appartenance à la communauté. Ensuite, la probabilité d'un bord est modélisée en fonction des affiliations de communautés partagées.
6. **HYCOM(Araujo et al., 2014)** :est un algorithme qui détecte les communautés à structure hyperbolique. Il se rapproche de la solution optimale en détectant de manière itérative les communautés importantes. L'idée clé est de trouver dans chaque étape une communauté unique qui minimise une fonction d'objectif basée sur le principe MDL en fonction des communautés précédemment détectées.

Nous présentons dans le tableau 2.3 une synthèse entre ces méthodes dans le but de mieux comprendre leurs impact sur les méthodes de compression. Nous constatons que le choix de l'algorithme de clustering peut orienter le processus de recherche vers un ou plusieurs types des structures ce qui influence le résultat de la compression.

	Chevauchement	Clique	Étoile	sous-graphe Bipartie	Chaîne	Structure Hyperbolique	Complexité
Metis	✗	Beaucoup	certaines	certains	peu	peu	O(m·k)
Spectral	✗	Beaucoup	certaines	Beaucoup	peu	peu	$O(n^3)$
Louvain	✗	Beaucoup	certaines	peu	peu	peu	$O(nlogn)$
SlashBurn	✓	Beaucoup	Beaucoup	certains	peu	peu	$O(t(m+nlogn))$
Bigclam	✓	Beaucoup	certaines	peu	peu	peu	O(d · n · t)
Hycom	✓	certaines	Beaucoup	certains	peu	Beaucoup	$O(k(m+hlogh^2+hm_h))$
KCBC	✓	Beaucoup	certaines	peu	peu	peu	O(t(m + n))

TABLE 2.3 – Tableau comparative entre les méthodes de clustering avec n = nombre de nœuds, m = nombre d'arêtes, k = nombre de clusters, t = nombre d'itérations, d = degré moyen de nœuds, h(m_h) = nombre de nœuds (arêtes) dans la structure hyperbolique.

Une première technique de compression usant de ces méthodes s'intitule VOG (Koutra et al., 2015). C'est une méthode de base sur laquelle s'appuient toutes les autres méthodes de cette classe. Elle permet de compresser un graphe statique non orienté G à l'aide d'un vocabulaire de sous-structures qui apparaissent fréquemment dans les graphes réels et ayant une signification sémantique, tout en minimisant le cout du codage en utilisant le principe de longueur de description minimale (MDL) [5]. Le vocabulaire Ω utilisé est composé de six structures qui sont : clique (fc) et quasi-clique (nc), noyau bipartie (cb) et quasi-noyau bipartie (nb), étoile (st) et chaine (ch). On peut avoir un chevauchement au niveau des nœuds, les liens quand à eux sont pris selon un ordre FIFO et ne peuvent pas se chevauchés, i.e 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 possibles 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 cout de stockage du modèle et de la matrice d'adjacence. En d'autre terme, VoG formule le problème de compression comme un problème d'optimisation dont la fonction objective est : $min(D,M)$ = L(M) + L(E) avec E= A $\bigoplus$ **M** représentant l'erreur.

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

$$\mathrm{L}(M) = L_{\mathbb{N}}(|\mathrm{M}|+1) + \log \begin{pmatrix} |M| + |\Omega\| - 1 \\ |\Omega| - 1 \end{pmatrix} + \sum_{s \in M} (-logPr(x(s)|M) + L(s))$$

Le premier terme représente le nombre de structures dans le modèle avec , le second terme encode le nombre de structures par type $x \in \Omega$ tant dis que le troisième terme permet pour chaque structure $s \in M$, d'encoder son type x(s) avec un code de préfixe optimale et d'encoder sa structure. Le codage des structures se fait selon leurs types :

Clique : Pour l'encodage d'une clique, on calcule le nombre de nœuds de celle-ci, et on encode leurs ids : L(fc) = $L_{\mathbb{N}}(|fc|)$ + log $\binom{n}{|fc|}$

5. MDL est un concept de la théorie de l'information permettant de trouver le modèle ayant une longueur minimale : $min(D,M)$ = L(M) + L($D \mid M$) où L(M) est la longueur du modèle et L($D \mid M$) est la longueur en bits de la description des donnés en utilisant le modèle M.

Quasi-Clique : Les quasi cliques sont encodées comme des cliques complètes, toute en identifiant les arêtes ajoutées dont le nombre est $||nc||$ et manquantes dont le nombre est noté $||nc||$' en utilisant des codes de préfixe optimaux : L(nc) = $L_{\mathbb{N}}(|nc|)$ + log $\binom{n}{|nc|}$ + log($|nc|$) + $||nc||l_1$ + $||nc||'l_0$ où l_1 = - log($||nc||/(||nc||+||nc||')$) et analogique 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 ensembles du noyau bipartie. On encode leurs tailles, ainsi que les identifiants 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 quasi-noyaux bipartie sont codés comme suit : L(nb) = $L_{\mathbb{N}}(|A|)$ + $L_{\mathbb{N}}(|B|)$ + log $\binom{n}{|A|}$ + log $\binom{n-|A|}{|B|}$+log(|area(nb)|) + $||nb||l_1$ + $||nb||'l_0$.

Étoile : L'étoile est un cas particulier d'un noyau bipartie. 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) = $L_{\mathbb{N}}(|st|\text{-}1)$ + log n + log $\binom{n-1}{|st|-1}$.

Chaine : On calcule d'abord le nombre d'éléments de la chaine, ensuite on encode les identifiants des nœuds selon leurs ordre dans la chaine : L(ch) = $L_{\mathbb{N}}(|ch|\text{ - }1) + \sum_{i=0}^{|ch|}(n-i)$

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

$$\text{L}(E^+) = \log(|E^+|) + ||E^+||l_1 + ||E^+||'l_0$$
$$\text{L}(E^-) = \log(|E^-|) + ||E^-||l_1 + ||E^-||'l_0$$

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 utilisées pour décomposer le graphe en sous-graphes pas forcément disjoints. La méthode de décomposition utilisée 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 pour une similarité sans erreur par rapport aux structures complètes du vocabulaire :
        - si tous les sommets d'un sous graphe d'ordre n ont un degré égale à n-1, il s'agit alors d'une clique.
        - si tous les sommets ont un degré de 2 sauf deux sommets ayant le degré 1, le sous-graphe est une chaine.

- si les amplitudes de ses valeurs propres maximales et minimales sont égaux, le sous-graphe est un noyau bipartie où les sommet de chaque nœuds sont identifiés à travers un parcours BFS avec 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 seule sommet.

b **Étiquetage des structures approximatives** : Si le sous graphe ne correspond pas à une structure complète, on cherche la structure qui l'approxime le mieux en terme du principe MDL.

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

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 sélections sont utilisées car le nombre de permutations est très grand ce qui implique des calcules exhaustifs. Les heuristiques permettent d'avoir des résultats approximatives et rapides, parmi les heuristiques utilisées dans VOG on trouve :
   — PLAIN : Cette heuristique retourne toutes 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 $\mathcal{C}$ ordonné selon la qualité (gain en bits), ajoute la structure au modèle tant qu'elle n'augmente pas le cout total de la représentation, sinon l'ignore.

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(Koutra et al., 2015) ne suppose que le premiers cas. En partant de ce constat, les auteurs de (Liu et al., 2015) proposent VoG-overlapp, une extension de VoG prenant en compte les chevauchement des structures sous forme d'une étude expérimentale de l'effet de diverses méthodes de décomposition sur la qualité de la compression.

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 :

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

Le principe de calcul de $L(M)$ *et* $L(E)$ demeurent le même, avec 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 (2.1).

$$L(O) = log(|O|) \ + \ ||O||\ l_1 \ + \ ||O||'\ l_0 \ + \ \sum_{o \in \varepsilon(O)} L_N(|o|) \tag{2.1}$$

Où :

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

Durant la même année, Shah et al.(Shah et al., 2015) 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 $G = \bigcup_{t_i} G_{t_i}(V, E_{t_i}) \quad 1 \preceq i \preceq t$ 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.

$$L(M) = L_N(|M|+1) + \log\tbinom{|M|+|\Phi|-1}{|\Phi|-1} + \sum_{s \in M}(logP(v(s)|M) + L(c(s)) + \mathbf{L(u(s))})$$

Le cout de l'encodage de la présence temporelle diffère 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_{debut} et t_{fin}. Le cout englobe le nombre de timestamps dans lesquels elle apparait ainsi que les identifiants des deux timestamp t_{debut} et t_{fin} : $L(u(s)) = L_N(|u(s)|) + log\ \binom{t}{2}$.
- **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 timestamps de manière aléatoire. Le coût doit englober donc le nombre de timestamps ainsi que leurs identifiants d'où : $L(u(s)) = L_N(|u(s)|) + log\ \binom{t}{|u(s)|}$.
- **Constant** : dans ce cas la sous-structure apparait dans tout les timestamps 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 les auteurs tirent parti du principe 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 dans ce cas l'erreur de l'encodage de la connectivité ainsi que l'erreur de l'encodage de la signature temporelle.

Algorithm 1 TIMECRUNCH

1: **Génération de sous-structure candidate :** Génération de sous-graphe pour chaque G_{t_i} en utilisant un des algorithmes de décomposition de graphe statiques
2: **Étiquetage de sous-structure candidate :** Associer chaque sous-structure à une etiquette $x \in \Omega$ minimisant son MDL.
3: **Assemblage des sous-structures candidates temporelles :** Assembler les sous-structure des graphes G_{t_i} pour former des structures temporelles avec un comportement de connectivité cohérente et étiquetez-les conformément en minimisant le coût de codage de la présence temporelle. Enregistrer le jeu de candidats $C_x \in C$.
4: **Composition du graphe compressé :** Composition du modèle M d'importantes structures temporelles non redondantes qui résument G à l'aide des méthodes heuristiques VANILLA, TOP-10, TOP-100 et STEPWISE. Choisir M associé à l'heuristique qui génère le coût de codage total le plus faible.

Une dernière variante a été présenté par Liu et al. (Liu et al., 2018b) où ils abordent efficacement trois contraintes principales des méthodes précédentes : (i) leurs dépendance à la méthode d'extraction de motifs (ii) l'incapacité de certaines à gérer les motifs qui se chevauchant (iii) leurs dépendance vis-à-vis de l'ordre dans lequel les structures candidates sont considérées lors de la phase d'assemblage. En effet, pour résoudre le premiers problèmes ils combinent plusieurs méthodes d'extraction de motifs ce qui améliore la qualité des structures candidates en dépit du temps d'exécution. Tant dis que pour répondre à la deuxième contrainte ils utilisent la fonction objective proposée dans (Liu et al., 2015). Arrivant à la dernière phase de l'algorithme **??**, ils proposent quatre nouvelles heuristique : (1) STEP : choisie les K meilleurs structures, (2) STEP-P : partitionne le graphe et affecte chaque motifs à la partition ayant un chevauchement maximal de noeuds avec lui. Ces partition sont parcourue parallelement pour ne prendre que meilleur de toutes les structures dans chacune des partitions, (3) STEP-PA :amélioration de STEP-P en désignant chaque partition du graphe comme étant active, puis si une partition échoue x fois pour trouver une structure qui réduit le coût MDL, cette partition est déclarée inactive et n'est pas visitée dans les prochaines itérations, (4) K-STEP : combinaison des trois premières heuristiques. Ils transforme par la suite chaque motif trouvé en un super-nœud.

Basée sur les propriétés de la matrice d'adjacence

Les graphes peuvent avoir différentes représentations. Chacune des structures de données présente des avantages et des inconvénients en ce qui concerne la quantité de mémoire nécessaire pour stocker les données et la facilité d'accès aux données. Selon les besoins, il est parfois utile de stocker les données dans des structures de données plus grandes, qui nécessitent plus d'espace mais offrent un accès efficace aux données. En se basant sur ce constat plusieurs méthodes ont été proposée dans la littérature pour compresser la matrice d'adjacence en exploitant les propriétés des graphes réels pour trouver les motifs les plus fréquents dans cette dernière.

(Asano et al., 2008) ont exploité les propriétés du graphe du web pour présenter une nouvelle méthode de compression (ECWG) sans perte permettant d'extraire les motifs à partir de la matrice d'adjacence. Il proposent un vocabulaire composée de six types de blocs (Motifs) : un bloc horizontale de 1, un bloc verticale de 1, un bloc diagonale de 1, un rectangle de 1, un bloc de 1 sous forme de L et le singleton 1. Avant de procédé à l'extraction des motifs, la liste d'adjacence du graphe est partitionner selon les hôtes. Une nouvelle matrice d'adjacence est donc construite pour chaque hôte contenant les liens existants entre ses pages aux quel les liens inter-hote sont concaténés. Les blocs B sont détectés par la suite et chacun est représenté par un quadruplets (i, j, type(B), dim(B)) où i,j représentent les coordonnées du premier élément du blocs dans la matrice d'adjacence de l'hôte, type(B) représente le type du bloc et dim(B) représente les dimensions du bloc (omis dans le cas du singleton).

Source	Destinations
0	1, 2, 315
1	0, 2, 3, 315, 316
2	3
3	1
4	314
...	...
314	4
315	0, 316, 317
316	1, 317
317	316

Source	Intra	Inter
315	316, 317	0
316	317	1
317	316	-

Source	Destinations
0	1, 2
1	2
2	1

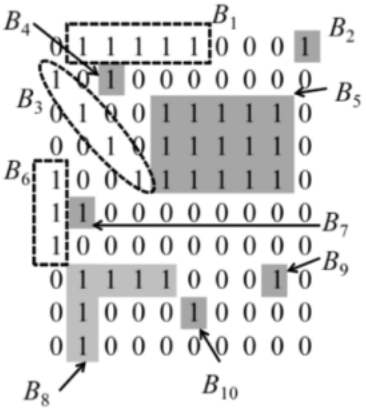

FIGURE 2.16 – Exemple illustrant le principe de fonctionnement (Asano et al., 2008)

Dans une méthode toute récente(GCUPMT), Shah et Rushabh (Shah, 2018) partitionnent les lignes de la matrice d'adjacence en plusieurs blocs ayant la même taille des motifs qui sont dans ce cas sous forme de vecteur prédéfinies. Les blocs sont comparés avec l'ensemble des motifs ce qui entraîne , en cas de correspondance , le remplacement du bloc par un indicateur du motif précédé par un 1 indiquant ainsi que les bits suivants appartenant à un indicateur de motif. Dans le cas contraire, les données brutes sont stockées directement précédés par un 0.

2.4.2 Compression basée Agrégation des motifs

Les méthodes de compression par extraction de motifs basées sur l'agrégation sont des méthodes qui agrègent plusieurs nœuds ou liens d'un motif en un seul nœud ou lien, appelés respectivement super-nœud et super-lien. Le graphe en sortie, dit super-graphe, devient dès lors plus simple et moins complexe offrant ainsi une aisance et une facilité de traitement, d'exploration et de visualisation. Nous présenterons dans ce qui suit les deux sous-classes de cette classe.

Compression basée Agrégation de nœuds

Les techniques de compression basées sur l'agrégation des nœuds des motifs sont des méthodes qui ont existé depuis plusieurs décennies offrant plusieurs avantages. Elles visent à résumer le graphe initial en agrégeant les nœuds des motifs découvert dans le but de diminuer le nombre de nœuds existants et d'offrir une meilleur visibilité et analyse du graphe.

Une première méthode de cette classe s'intitulent Subdue (Ketkar et al., 2005). Elle effectue une recherche *Branch&Bound* qui commence à partir de sous-structures composées de tous les sommets avec des étiquettes uniques. Les sous-structures sont prolongées de toutes les manières possibles par un sommet et une arête ou par une arête afin de générer des sous-structures candidates. Subdue conserve les instances de sous-structures et utilise l'isomorphisme de graphe pour déterminer les instances de la sous-structure candidate. Les sous-structures sont ensuite évaluées en fonction de leur compression de la longueur de description (DL) du jeu de données. Cette procédure se répète jusqu'à ce que toutes les sous-structures soient prises en compte ou que les contraintes imposées par l'utilisateur ne soient plus vérifiés. A la fin de la procédure, Subdue indique les meilleures sous-structures de compression. Le système Subdue fournit également la possibilité d'utiliser la meilleure sous-structure trouvée lors d'une étape de découverte pour compresser le graphe d'entrée en remplaçant ces instances de la sous-structure par un seul sommet et en effectuant le processus de découverte sur le compressé. Cette fonctionnalité génère une description hiérarchique du jeu de données de graphe à différents niveaux d'abstraction en termes de sous-structures découvertes.

(Rossi and Zhou, 2018) partent de l'observation que graphes réels sont formés souvent de nombreuses cliques de grande taille. En utilisant ceci comme base, GraphZip décompose le graphe en un ensemble de grandes cliques, qui est ensuite utilisé pour compresser et représenter le graphe de manière succincte.

Compression basée agrégation de liens

Les méthodes de compression par extraction de motifs basées agrégation de liens sont parmi les méthode les plus populaires. Leur objectif est de produire un graphe compressé à partir du graphe initial en remplaçant les liens denses du graphe par un nouveau super-nœud ou une nouvelle super-arêtes. Elle se divise selon le principe en deux grandes classes : celles utilisant les règles de grammaire et celles utilisant des méthode de clustering. Nous

détaillerons dans ce qui suit ces deux classes et nous conclurons par une synthèse sur les méthodes de chaque classe.

Basées sur les règles de grammaire

La classe des méthodes de compression basées sur les règles de grammaire est une généralisation d'une méthode de compression des dictionnaire s'intitulant Re-pair. Son principe de base consiste en la recherche, à chaque itération, de la paire de symboles la plus fréquente dans une séquence de caractères et à la remplacer par un nouveau symbole, jusqu'à ce qu'il ne soit plus commode de les remplacer. Nous notons que dans ce cas le motif est sous forme de deux arêtes ayant un sommet en commun, nommé *digraph*.

Une première méthode de suivant ce principe a été proposée dans (Claude and Navarro, 2010b) baptisé *Approximate Re-pair*. Dans cette méthode un graphe G=(V,E) est représenté sous forme d'une sequence de caractères T :

$$\mathrm{T}=\mathrm{T}(\mathrm{G})=\ \overline{v_1}\ v_{1,1}\ \ldots\ v_{1,a}\ \overline{v_2}\ v_{2,1}\ \ldots\ v_{2,a_2}\ldots\overline{v_n}\ v_{n,1}\ \ldots\ v_{n,a_n}$$

où $\overline{v_i}$ représente un indicateur du sommet v_i. Elle procède en trois étapes essentielles expliquer dans l'algorithme **2** . Lorsqu'il n'y a plus de paires à remplacer, Approximate

Algorithm 2 Approximate Re-pair

1: **Calcule des fréquences :** T est parcourue séquentiellement et chaque pair $t_i t_{i+1}$ est ajouté à un tableau de hachage H avec leurs nombre d'occurence.
2: **Recherche des k meilleurs paires :** H est parcourue et les k paires les plus fréquentes sont retenues, en utilisant k pointeurs vers les cellules de H.
3: **Le remplacement simultané :** les k paires identifiés dans l'étape précédente sont simultanément remplacées par un nouveau identifiant et une règle de production est ajoutée.

Re-pair s'arrête donnant comme résultat un compressé compacte C de la chaine T. Pour finaliser le processus, tous les indicateurs de nœuds $\overline{v_i}$ seront supprimés de C. De plus, l'algorithme crée une table qui contiendra des pointeurs vers le début de la liste d'adjacence de chaque nœud dans C. Grâce à cette table l'algorithme pourra répondre aux requêtes de recherche de successeurs en un temps optimal.

Dans un travail ultérieure (Claude and Navarro, 2010a) , les même auteurs s'intéressent aux requêtes de recherche des nœuds prédécesseurs et successeurs à partir du graphe compressé de *Approximate Re-pair* directement. Il proposent alors de combiner leurs méthode avec une représentation basé sur les relations binaires de (Barbay et al., 2006). En effet, ce dernier consiste à représenter les listes d'adjacence à l'aide d'une représentation séquentielle permettant de rechercher les occurrences d'un symbole puis de rechercher les voisins inverses à l'aide de cette primitive.

Claude et Ladra (Claude and Ladra, 2011) partageaient les même préoccupations des auteurs de la méthode précédente et ont proposé comme solution de combiner la méthode Re-pair avec la représentation k2-tree. Ils obtiennent alors une compression de 2,27 (pbe) sur le graphe UK2002, tout en conservant la possibilité d'interroger les voisins entrants et sortants (Maneth and Peternek, 2015).

Une dernière méthode de cette classe s'instituant gRepair a été proposé dans (Maneth and Peternek, 2018). Ce nouveau algorithme de compression détecte de manière récursive des sous-structures répétées et les représente via des règles de grammaire. Des requêtes spécifiques telles que l'accessibilité entre deux nœuds ou des requêtes de chemin normal peuvent ainsi être évaluées en temps linéaire (ou en temps quadratiques, respectivement), sur la grammaire, permettant ainsi des accélérations proportionnelles au taux de compression. la figure 2.17 présentent le résultat de cette méthode sur un exemple.

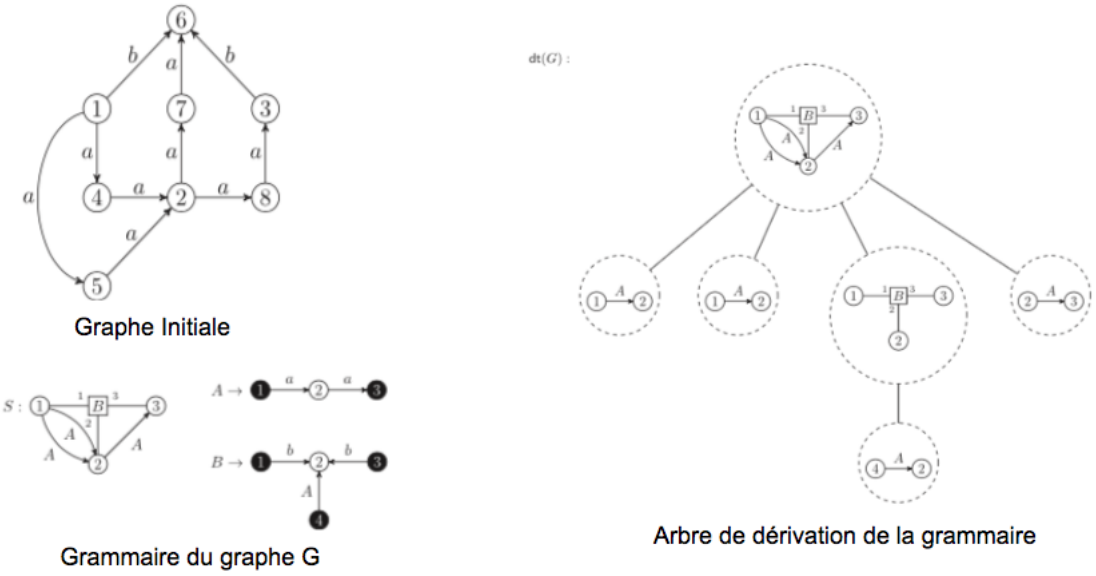

FIGURE 2.17 – Exemple d'exécution de gRepair sur G.

Basées sur des méthodes de clustering

Les méthodes de compression appartenant à la classe courante sont des méthodes basées sur la recherche des sous-graphes denses (ayant des nœuds fortement connectés). Ils sont destinées principalement aux graphes du Web et les graphes des réseaux sociaux dans but de faciliter leurs exploration et analyse.

(Buehrer and Chellapilla, 2008) ont exploité l'existence de plusieurs ensembles de pages web qui ont les même liens sortants. S'intitulant VNM pour Virtual Node Miner, leurs approche est basée sur la réduction du nombre de liens en créant des nouveaux sommets virtuels qui sont ajoutés au graphe. Soit G = (V,E) un graphe orienté, l'algorithme proposé se compose de deux phases essentielles :

1. **Phase de Clustering :**

   Le but de cette première étape est de contourner la tâche presque impossible d'extraction simultanée de centaines de millions de points de données en groupant d'abord les sommets similaires dans le graphes dans des clusters. Pour cela k fonctions de hachage indépendantes sont utilisées pour obtenir une matrice de taille V * k. Par la suite, les lignes de la matrice sont triées lexicographiquement et elle est parcourue colonne par colonne en regroupant les lignes ayant la même valeur. Lorsque le nombre total de lignes chute au-dessous d'un seuil ou que le bord de la

matrice de hachage est atteint, les identifiants des sommets associés aux lignes sont renvoyé au processus d'extraction (Phase 02).

2. **Phase d'Extraction de Motifs :** Le but de cette étape est de localiser des sous-ensembles communs de liens sortants dans les sommets donnés. Ainsi les ensembles plus grands et fréquents présentent un intérêt, car ils peuvent représenter des motifs plus pertinents et une meilleure compression. En effet, les performances de compression d'un motif sont calculés en fonction de sa fréquence dans la liste d'adjacence, et de sa taille qui est le nombre de liens qu'il contient (2.2).

$$Compression(P) = (P.frquence - 1)(P.taille - 1) - 1 \tag{2.2}$$

   Afin d'extraire ces motifs, VNM utilise une heuristique gloutonne. Cette heuristique procède comme suit :

   (a) Extraire un histogramme des identifiants de liaison sortante à partir de la liste d'adjacence des sommets données.

   (b) Les listes sont réorganisées dans l'ordre décroissant des fréquences des liens sortants en éliminant ceux qui apparaissent une seul fois uniquement.

   (c) Chaque lien sortant est ajouté à un arbre de préfixes avec l'ensemble trié de ces extrémités initiales selon leurs identifiants.

   (d) L'arbre est par la suite parcouru afin d'identifier les motifs qui maximisent la formule de performance de la compression 2.2. Ces motifs sont ensuite convertis en nœuds virtuels et les identificateurs de sommet de leurs listes sont supprimés.

L'algorithme est appliqué jusqu'à ce que la réduction n'apporte pas un gain significative. La figure 2.18 illustre le principe de fonctionnement de cette méthode sur un exemple.

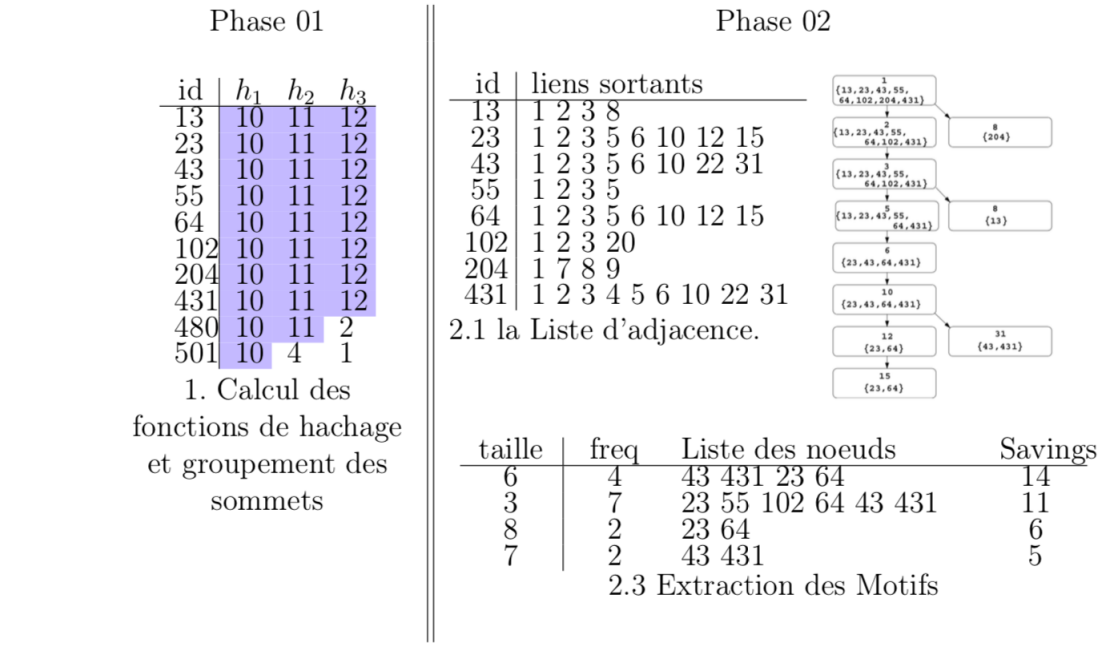

FIGURE 2.18 – Exemple d'exécution de VNM.

Une variante de VNM a été proposée par Hernandez et Navarro (Hernández and Navarro, 2014). Comme première contribution, ils augmentent les types de structures découvertes dans la phase de clustering pour englober aussi : les cliques, les bi-cliques. L'extraction de motifs cette fois-ci n'est basée sur un parcours des feuilles vers la racine mais l'inverse où l'ensemble des sommets finales des liens du motifs est constitués des étiquettes des nœuds de l'arbre inclus dans le chemin de la racine vers la feuille et les sommets initiales sont la liste des sommets inclus dans le nœud feuille. Leurs deuxième contribution consiste en une hybridation dans le but de représenter le graphe en sortie à l'aide de structures compactes. Une première approche proposée est d'utiliser les k2-trees (Brisaboa et al., 2009) et qui donnent la représentation la plus compacte. La deuxième hybridation consiste en une nouvelle structure proposée par les auteur.

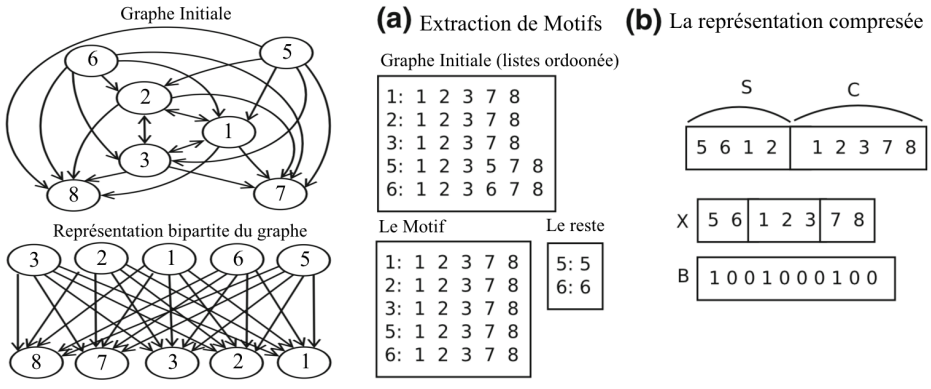

FIGURE 2.19 – Exemple d'exécution de SDM

Classe	Méthode	Graphe en entrée				Compression		Structure en sortie		Graphe de test	Résultat
		Orienté	Non orienté	Statique	Dynamique	Avec perte	Sans perte	Succincte	Structurelle		
Basée vocabulaire en utilisant les méthodes de clustering	VoG (Koutra et al., 2015)	✗	✓	✓	✗	✗	✓	✓	✗	ASOregon	71%
	VoG-Overlapp (Liu et al., 2015)	✗	✓	✓	✗	✗	✓	✓	✗	ASOregon	25%
	TimeCrunch (Shah et al., 2015)	✗	✓	✓	✗	✗	✓	✓	✗	Enron	74%
	CanDenSe (Liu et al., 2018b)	✗	✓	✗	✓	✗	✓	✓	✓	Enron	78%
Basée vocabulaire en utilisant les propriétés de la matrice d'adjacence	ECWG (Asano et al., 2008)	✓	✓	✓	✗	✗	✓	✓	✗	uk-2002	76.1%
	GCUPMT (Shah, 2018)	✓	✓	✓	✗	✗	✓	✓	✗	graphes 8192 nœuds	70%
Basée Agrégation de nœuds des motifs	Subdue (Ketkar et al., 2005)	✗	✓	✓	✗	✗	✓	✗	✓	Graphe des composantes chimiques	16%
	GraphZip (Rossi and Zhou, 2018)	✗	✓	✓	✗	✗	✓	✗	✓	Web-Google	19%
Basée Agrégation de liens des motifs en utilisant les règles de grammaire	Approximate Re-pair (Claude and Navarro, 2010b)	✓	✗	✓	✗	✗	✓	✓	✗	uk-2002	4.23
	Approximate Re-pair (Claude and Navarro, 2010a)	✓	✗	✓	✗	✗	✓	✓	✗	uk-2002	3.98
	gRe-pair (Maneth and Peternek, 2018)	✓	✓	✓	✗	✗	✓	✓	✗	NotreDame	4.84
Basée Agrégation de liens des motifs et les méthodes de clustering	VNM (Buehrer and Chellapilla, 2008)	✓	✗	✓	✗	✗	✓	✗	✓	uk-2002	1.95
	DSM (Hernández and Navarro, 2014)	✓	✗	✓	✗	✗	✓	✓	✓	uk-2002	1.53

TABLE 2.4 – Synthèse des méthodes de compression par extraction de motifs.

2.5 Bilan générale

Nous avons étudiés dans ce chapitre deux classes de compression de graphes : (i) Les méthodes de compression par les k2-trees, (ii) Les méthodes de compression par extraction de motifs. Nous avons ainsi présenter le principe de fonctionnement des méthodes de compression relatifs à chaque classe tout en comparant entre leurs caractéristiques.

Cependant une étude comparative entre les deux classes reste primordiale. Dans le tableau 2.5 , nous allons essayer de synthétiser les principaux différences et similitudes entre ces deux classes que nous avons put constater à travers notre recherche bibliographique et qui ont un fort impact sur le choix de la méthode de compression.

	k^2-trees	**Extraction de motifs**
Type de compression	toujours sans perte	toujours sans perte
Structure en sortie	Toujours succincte	Peut être succincte où structurelle où les deux en même temps
technique utilisée	exploitation de la matrice d'adjacence du graphe	exploitation des motifs fréquents dans le graphe
Dépendance	Dépend du paramètre k	Dépend selon la méthode de l'algorithme de clustering ou du vocabulaire de motifs utilisé
Objectif	Compression, Réduire l'espace de stockage et le temps de parcours	- Compression, - Réduire l'espace de stockage et le temps de parcours, - Extraire les informations pertinentes, - Visualisation
Domaine d'application	tous les domaines	tous les domaines

TABLE 2.5 – Comparaison entre les méthodes basées surk^2-trees et basées sur l'extraction de motifs.

Chapitre 3

Contribution

Comme on vient de voir dans les chapitres précédents, de nombreux phénomènes dans des contextes très variés peuvent être représenter par des graphes. Citons en particuliers les réseaux sociaux qui nous attirent vue leurs évolutions. Aujourd'hui 3.0 milliards de personnes utilisent les réseaux sociaux, voir 40% de la population sur terre. Ces grandes tailles des données manipulées rendent impossible tout traitement et demandent un grand espace de stockage faisant intervenir des algorithmes de compression adéquats qui changent selon le type de graphes. Les graphes statiques ne sont pas toujours suffisants pour décrire les situations de la vie réel et leurs évolutions dans le temps. Ajouter une dimension temporelle a ces graphes permet d'élargir l'espace de possibilités, les réseaux sociaux en ligne est un exemple qui nous captive où le suivi dynamique du graphe est nécessaire. Quoiqu'il existe de nombreuses techniques efficaces adaptés au contexte statiques, rares sont ceux qui proposent une contrepartie dynamique, ce qui nous amène à nous intéresser particulièrement à ce type de graphes toujours en cours d'évolutions. Nous nous sommes intéressés aussi dans ce travail à un autre enjeu important qui provient de la structures complexe de ces graphes qui pose un problème lors de l'exploration du graphe ce qui rend difficile l'analyser des données et l'extraction des informations pertinentes. La recherche de motifs et particulièrement les clique et les noyaux bipartie qui sont fréquents dans les réseaux sociaux répond à ce problème.
Pour résumé, Nous allons dans cette partie proposé une nouvelle méthode DDSM basé sur deux approches existantes que nous avons étudier dans notre état de l'art. Notre but principal est de développer une technique de compression compétitives pour les graphes des réseaux sociaux dans un contexte dynamique en se basant sur l'extraction de motifs. Pour accomplir ce travail nous nous somme appuyer sur ces deux méthodes :

— Exploiter la structure de donnés compressés proposé dans DSM (Hernández and Navarro, 2014) qui permet de représenter les sous-graphes denses comme les cliques, les bi-cliques, noyaux bipartis. Nous avons optés pour cette structure car on l'appliquant sur les réseaux sociaux, elle donne de bons résultats en termes d'espace et de temps de requêtes.

— Adopter les signatures temporelles utilisés dans TimeCrunch (Shah et al., 2015) pour décrire le comportement des sous-graphes dans le temps. Nous avons utilisé ces signatures car elle englobent tous les types des sous graphes temporelles qu'on peut trouver dans un graphe dynamique et c'est les seules qu'on a rencontrer dans ce genre.

Nous allons dans cette partie présenter notre méthode théoriquement : principe générale, structures utilisés et un pseudo code.

3.1 Formulation du problème

Dans cette section, nous allons définir le problème de base auquel notre méthode convient tout en définissant le cadre dans lequel elle peut être appliquée.

Nous considèrent les graphes orientés dynamiques $G = \bigcup_{t_i} G_{t_i}(V, E_{t_i})$ $1 \preceq i \preceq t$ où G_{t_i} = G a l'instant t_i et un ensemble de signature temporelle. En d'autre termes, nous considérons des captures du graphe à des instants différents t_i et un ensemble de descripteurs de comportement temporels des sous graphes des différents G_{t_i}. Le problème peut ainsi être formulé :

Problème : *Trouver, à partir d'un graphe dynamique G et un lexique de signatures temporelles, la plus petite description du graphe initiale en terme de ces structures les plus denses et leur comportement temporel, en offrant la possibilité d'extraire les voisins directes et d'extraire les sous-graphes au besoin.*

3.2 Principe générale

Notre méthode s'intitule DDSM pour *Dynamic Dense Substructure Mining*. Elle représente une généralisation du travail de Hernandez et Navarro(Hernández and Navarro, 2014) pour le cas des graphes dynamiques qui sont de nos jours omniprésents dans différents domaines.

Pour mieux expliquer notre méthode nous allons tout d'abords commencer par décrire les structures que nous allons utiliser. Nous enchainons par la suite avec le pseudo algorithme détaillant les étapes essentielles de notre méthode et nous conclurons en expliquant comment les algorithmes proposés dans (Hernández and Navarro, 2014) de manipulation des graphes peuvent être généraliser dans notre cas.

3.2.1 Description conceptuelle

La codification du graphe en sortie doit respecter les contraintes du problème tout en réduisant un maximum d'espace mémoire. Nous proposons pour cela d'augmenter la codification proposées dans (Hernández and Navarro, 2014) dans le cas des graphes dynamiques en rajoutant une l'information temporelle.

Une fois les sous-structures identifiées, hernandez et al. (Hernández and Navarro, 2014) proposent de représenter chacune d'elles avec trois composantes : la première contenant les sommet ayant uniquement des aretes sortantes, la deuxième contenant les sommet ayant des arêtes entrantes et sortantes et la troisième contenant les sommet ayant uniquement des arêtes entrantes. Pour pouvoir identifier les différentes composantes, ils associent un vecteur binaire à cette représentation marquant par un 1 le début de chacune des trois composantes (voir figure 2.19).

Notre première contribution consiste en l'extension de cette structure. En effet, nous suggérons de representer chaque structure avec non trois composantes mais quartes. Les trois premières étant les même que dans (Hernández and Navarro, 2014), la quatrième représente l'information temporelle. Cette dernière peut être sous cinq formats (voir 2.4.1) dont nous résumons la représentation proposée pour chacune dans le tableau 3.2.1.

Nous représentons G donc en tant qu'un ensemble de sous-graphes temporels denses. Cependant, pour obtenir une compression sans perte, nous devons aussi garder information sur l'erreur modélisant l'ensemble d'arêtes restantes dans chaque G_{t_i}. Nous proposons pour cela d'utiliser

Signature temporelle	Représentation
constante	0
OneShot	1 t_1
ranged	2 t_1 t_2
periodic	3 T
flikering	4 t_1 t_2 ... t_n

TABLE 3.1 – Les types de signatures temporelles et leurs représentations, t_i représentent les timestamps et T représente la période.

une des structures dynamiques des k2-trees qui nous permettra d'interroger le graphe de manière simple et efficace.

3.2.2 Notre méthode : DDSM

Nous visons à travers la méthode que nous proposons d'exprimer le graphe en entrée avec ses sous-structures les plus dense et leurs comportement temporels réduisant ainsi sa taille et offrant la possibilité d'effectuer les traitements dans un temps meilleur. Nous avons structurer notre algorithme sous forme de trois (03) étapes essentielles, chacune servant d'entrée pour l'étape suivante.

En premier lieu, nous appliquons la découverte des sous-graphes les plus denses. Pour effectuer cela de manière efficace, nous suggérons d'utiliser l'approche proposée dans (Hernández and Navarro, 2014) parallèlement sur chaque capture G_{t_i} de G. Nous allons donc considérer les listes d'adjacences pour chaque G_{t_i} où des fonctions de hachages sont calculées pour chaque sommet. Une arborescence est ensuite construite pour chaque ensemble de sommets ayant les même valeurs de hachages et qui permettra d'extraire les structures denses du graphe à travers un parcours de la racine vers les feuilles.

Dans une deuxième phase, nous effectuons une comparaison entre les structures découvertes dans des timestamps différents. Nous proposons pour décrire le comportement de ces derniers d'utiliser le même ensemble de signatures temporelles proposées dans (Shah et al., 2015) composées de six descripteurs temporelles englobant tout les cas possibles. Durant cette étape, nous tolérons un certains seuil d'erreur qui sera inclus dans la représentation de l'erreur.

Une dernière phase consiste en la codification du graphe en utilisant le codage proposés dans la section précédentes pour chaque structure de a phase (02). Nous concaténons par la suite ces représentations pour obtenir une seule représentation concise du graphe initiale en terme de ses structures les plus denses. L'algorithme ci-dessous résume les trois principales étapes de notre méthode :

Algorithm 3 DDSM

1: **Génération des sous-structures candidates :** Génération de sous-graphes, principalement les sous-graphes bipartis et les cliques.
2: **Étiquetage de sous-structures :** Associer chaque sous-structure à une signature temporelle décrivant son comportement.
3: **Codification du graphe compressé :** Codifier les sous-structures étiquetées de l'étape 02 en utilisant les structures de la section précédente.

Pour les algorithmes de parcours et d'extraction de voisins, tous les algorithmes proposés dans (Hernández and Navarro, 2014) peuvent être appliqués sur notre structure. En effet, le seul changement consiste en la prise en compte de l'information temporelle dans l'incrémentation des indices de parcours. De ce fait, nous pensons que notre méthode peut offrir un très bon compromis entre l'espace mémoire et le temps d'accès des traitements dans le cas des graphes dynamiques.

3.3 Conclusion

Dans ce chapitre, nous avons abordés le problème de compression des graphe dynamique par extraction de motifs. Nous avons formaliser le problème de compression d'un graphe dynamique en se basant sur les sous-graphes denses temporelles qui le composent. Pour faire face a ce problème nous avons présenter une nouvelle méthode hybride intitulé DDSM, elle combine entre deux méthodes de la littérature que nous avons juger efficaces. Notre prochain but est de tester cette méthode en la comparant avec les méthodes existantes.

Conclusion générale

La situation d'abondance d'information aujourd'hui a favorisé l'incapacité d'assimilation et de réemploi de l'information et a montré l'inadaptation des outils de gestion classiques. Une des solution les plus connus pour pallier à ce problème est la compression.

Nous nous sommes intéressés dans ce travail aux domaines de compression de graphes, et plus particulièrement les méthodes de compression par les k2-trees et les méthodes de compression par extraction de motifs. Nous avons présenté les différentes méthodes de compression existantes et nous avons établi une étude comparatif entre eux. Nous avons aussi essayer de rectifier la classification des proposées dans un travail de master précédent. Une dernière contribution que nous avons apporter consiste en une nouvelle méthode de compression par extraction de motifs usant des structures k2-trees.

Cette étude nous a amené à constater que malgré le progrès existant dans ce domaine plusieurs défis, tel que :la compression temps réelle et les temps de traitements, sont encore présents. Ces défis constitues de nouvelles problématiques de recherche qui sont toujours ouvert.

Nous nous focaliserons dans le future à implémenter ces méthodes et leurs variantes. Cette implémentation nous permettra de mieux comparer entre ces méthodes en employant le même jeux de données. De ce fait nous pourrons établir une comparaison plus objective entre eux.

Bibliographie

(2012). *Quelques rappels sur la théorie des graphes*. IUT Lyon Informatique.

Alvarez-Garcia, S., de Bernardo, G., Brisaboa, N. R., and Navarro, G. (2017). A succinct data structure for self-indexing ternary relations. *Journal of Discrete Algorithms*, 43 :38–53.

Álvarez-García, S., Freire, B., Ladra, S., and Pedreira, Ó. (2018). Compact and efficient representation of general graph databases. *Knowledge and Information Systems*, pages 1–32.

Aragon, C. R. and Seidel, R. G. (1989). Randomized search trees. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 540–545. IEEE.

Araujo, M., Günnemann, S., Mateos, G., and Faloutsos, C. (2014). Beyond blocks : Hyperbolic community detection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 50–65. Springer.

Asano, Y., Miyawaki, Y., and Nishizeki, T. (2008). Efficient compression of web graphs. In *International Computing and Combinatorics Conference*, pages 1–11. Springer.

Badr, M. (2013). *Traitement de requêtes top-k multicritères et application à la recherche par le contenu dans les bases de données multimédia*. PhD thesis, Cergy-Pontoise.

Barbay, J., Golynski, A., Munro, J. I., and Rao, S. S. (2006). Adaptive searching in succinctly encoded binary relations and tree-structured documents. In *Annual Symposium on Combinatorial Pattern Matching*, pages 24–35. Springer.

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of statistical mechanics : theory and experiment*, 2008(10) :P10008.

Boldi, P. and Vigna, S. (2004). The webgraph framework i : compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602. ACM.

Brisaboa, N. R., de Bernardo, G., Gutiérrez, G., Ladra, S., Penabad, M. R., and Troncoso, B. A. (2015). Efficient set operations over k2-trees. In *Data Compression Conference (DCC), 2015*, pages 373–382. IEEE.

Brisaboa, N. R., De Bernardo, G., Konow, R., and Navarro, G. (2014a). K 2-treaps : Range top-k queries in compact space. In *International Symposium on String Processing and Information Retrieval*, pages 215–226. Springer.

Brisaboa, N. R., De Bernardo, G., and Navarro, G. (2012). Compressed dynamic binary relations. In *Data Compression Conference (DCC), 2012*, pages 52–61. IEEE.

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.

Brisaboa, N. R., Ladra, S., and Navarro, G. (2013). Dacs : Bringing direct access to variable-length codes. *Information Processing & Management*, 49(1) :392–404.

Brisaboa, N. R., Ladra, S., and Navarro, G. (2014b). Compact representation of web graphs with extended functionality. *Information Systems*, 39 :152–174.

Buehrer, G. and Chellapilla, K. (2008). A scalable pattern mining approach to web graph compression with communities. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pages 95–106. ACM.

Claude, F. and Ladra, S. (2011). Practical representations for web and social graphs. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1185–1190. ACM.

Claude, F. and Navarro, G. (2010a). Extended compact web graph representations. In *Algorithms and Applications*, pages 77–91. Springer.

Claude, F. and Navarro, G. (2010b). Fast and compact web graph representations. *ACM Transactions on the Web (TWEB)*, 4(4) :16.

De Berg, M., Van Kreveld, M., Overmars, M., and Schwarzkopf, O. (1997). Computational geometry. In *Computational geometry*, pages 1–17. Springer.

De Bernardo, G., Álvarez-García, S., Brisaboa, N. R., Navarro, G., and Pedreira, O. (2013). Compact querieable representations of raster data. In *International Symposium on String Processing and Information Retrieval*, pages 96–108. Springer.

de Bernardo Roca, G. (2014). *New data structures and algorithms for the efficient management of large spatial datasets*. PhD thesis, Citeseer.

Fages, J.-G. (2014). *Exploitation de structures de graphe en programmation par contraintes*. PhD thesis, Ecole des Mines de Nantes.

Garcia, S. A., Brisaboa, N. R., de Bernardo, G., and Navarro, G. (2014). Interleaved k2-tree : Indexing and navigating ternary relations. In *Data Compression Conference (DCC), 2014*, pages 342–351. IEEE.

Grossi, R., Gupta, A., and Vitter, J. S. (2003). High-order entropy-compressed text indexes. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 841–850. Society for Industrial and Applied Mathematics.

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.

Hernández, C. and Navarro, G. (2014). Compressed representations for web and social graphs. *Knowledge and information systems*, 40(2) :279–313.

Hespanha, J. P. (2004). An efficient matlab algorithm for graph partitioning. *Santa Barbara, CA, USA : University of California*.

Jean-Charles Régin, A. M. (2016). Théorie des graphes. Technical report.

Kang, U. and Faloutsos, C. (2011). Beyond'caveman communities' : Hubs and spokes for graph compression and mining. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 300–309. IEEE.

Karypis, G. and Kumar, V. (2000). Multilevel k-way hypergraph partitioning. *VLSI design*, 11(3) :285–300.

Ketkar, N. S., Holder, L. B., and Cook, D. J. (2005). Subdue : Compression-based frequent pattern discovery in graph data. In *Proceedings of the 1st international workshop on open source data mining : frequent pattern mining implementations*, pages 71–76. ACM.

Khan, K. U., Nawaz, W., and Lee, Y.-K. (2014). Set-based unified approach for attributed graph summarization. In *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*, pages 378–385. IEEE.

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.

LeFevre, K. and Terzi, E. (2010). Grass : Graph structure summarization. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 454–465. SIAM.

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., Safavi, T., Dighe, A., and Koutra, D. (2018a). Graph summarization methods and applications : A survey. *ACM Computing Surveys (CSUR)*, 51(3) :62.

Liu, Y., Safavi, T., Shah, N., and Koutra, D. (2018b). Reducing large graphs to small supergraphs : a unified approach. *Social Network Analysis and Mining*, 8(1) :17.

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.

Maneth, S. and Peternek, F. (2015). A survey on methods and systems for graph compression. *arXiv preprint arXiv :1504.00616*.

Maneth, S. and Peternek, F. (2018). Grammar-based graph compression. *Information Systems*, 76 :19–45.

Martínez-Prieto, M. A., Fernández, J. D., and Cánovas, R. (2012). Compression of rdf dictionaries. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 340–347. ACM.

Müller, D. (2012). *Introduction à la théorie des graphes*. Commission romande de mathématique (CRM).

Parlebas, P. (1972). Centralité et compacité d'un graphe. *Mathematiques 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 graphesorie des graphes*. Université de liège, Faculté des sciences Département de mathématiques.

Rossi, R. A. and Zhou, R. (2018). Graphzip : a clique-based sparse graph compression method. *Journal of Big Data*, 5(1) :10.

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.

Shah, R. J. (2018). Graph compression using pattern matching techniques. *arXiv preprint arXiv :1806.01504*.

Shen, Z., Ma, K.-L., and Eliassi-Rad, T. (2006). Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *IEEE transactions on visualization and computer graphics*, 12(6) :1427–1439.

Shi, L., Tong, H., Tang, J., and Lin, C. (2015). Vegas : Visual influence graph summarization on citation networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(12) :3417–3431.

Shi, Q., Xiao, Y., Bessis, N., Lu, Y., Chen, Y., and Hill, R. (2012). Optimizing k2 trees : A case for validating the maturity of network of practices. *Computers & Mathematics with Applications*, 63(2) :427–436.

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*.

W. GUERMAH, T. B. (2018). *Compression de Graphes : étude et classification*. PhD thesis, Esi.

Yang, J. and Leskovec, J. (2013). Overlapping community detection at scale : a nonnegative matrix factorization approach. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM.

Zhang, H., Duan, Y., Yuan, X., and Zhang, Y. (2014a). Assg : Adaptive structural summary for rdf graph data. In *International Semantic Web Conference (Posters & Demos)*, pages 233–236. Citeseer.

Zhang, Y., Xiong, G., Liu, Y., Liu, M., Liu, P., and Guo, L. (2014b). Delta-k 2-tree for compact representation of web graphs. In *Asia-Pacific Web Conference*, pages 270–281. Springer.