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)

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

---

Réaliser par :

Mlle. Hafsa Bousbiat
eh_bousbiat@esi.dz
ESI

Mlle. Sana Ihadadene
es_ihadadene@esi.dz
ESI

Encadreurs :

Dr. Karima Amrouche
k_amrouche@esi.dz
ESI

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

Dr. Mohammed Haddad
mail
Université de Lyon

Octobre 2018

Année Universitaire : 2018-2019

Remerciement

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

Résumé

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

Abstract

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

Table des matières

Table des figures

Liste des tableaux

Première partie

Introduction

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 cours des années au sein des autres disciplines comme la chimie, la biologie, la sociologie et l'industrie. Elle constitue aujourd'hui un corpus de connaissance très important et un instrument efficace pour résoudre une multitude de problèmes.
De manière général le graphe sert à représenter les structures, les connexions entre différents composants, les acheminements possible pour un ensemble complexe composé d'un grand nombre de situations, en exprimant les dépendances et les relations entre ses éléments,(e.g. réseau routier ou ferroviaire, réseau de communication,diagramme d'ordonnancement, ..).
Dans ce chapitre nous introduisons les définitions et notions de base relatives aux graphes que nous utiliserons par la suite.

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 $\{ v_i, v_j \} \in E \subset V \times V$ 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 représenter par des points distincts.

— Les arêtes de G : $e = (v_i, v_i) \in$ E sont représenter 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 E={ (1,2), (1,4), (2,2), (2,3) ,(2,5) ,(3,4) }. La représentation graphique de g est alors donnée

par le schéma de la figure 1.1.

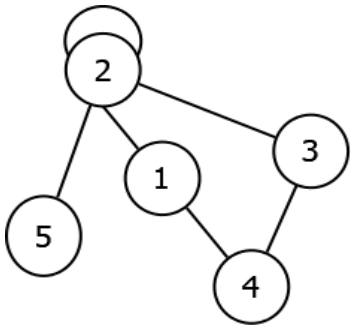

FIGURE 1.1 – Exemple de représentation graphique d'un graphe non orienté

1.1.3 Propriété 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 d'un graphe :

Degré d'un sommet : Le degré d'un sommet noté $d(v_i)$ est le nombre d'arêtes incidents a ce sommet, sachant qu'une boucle compte pour 2 (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. e.i. 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 diamétre d'un graphe :

Distance : La distance entre deux sommets v et u est le plus petit nombre d'arêtes qu'on doit parcourir pour aller de v a u ou de u a 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 pette 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 graphe).

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

— v_i est dit extrémité initiale ou origine de e et v_j est l'extrémité finale de e (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 donnée par le schéma de la figure ci-dessous.

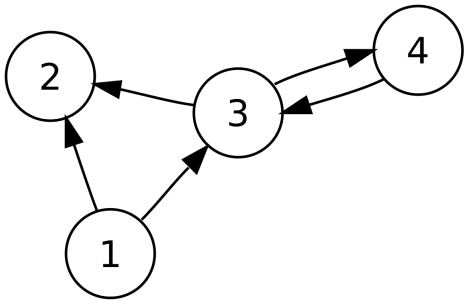

FIGURE 1.2 – Exemple de représentation graphique d'un digraphe.

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 :

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

avec succ(v_i) qui est l'ensemble des successeurs de v_i et pred(v_i) qui 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 Quelques types de graphe

Avec les avancées technologique 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 tout pair de sommet est reliée par au plus une arête. Dans le cas contraire, G est dit multiple (IUT, 2012).
- **Graphe connexe :** Un graphe non orienté (resp. orienté) est dit connexe (resp. fortement connexe) si pour tout pair de sommets (v_i, v_j) il existe un chemin S les reliant (Müller, 2012).

1.4 Graphe partiel et sous graphe :

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

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 : V' = $V_1 \cup V_2$, tel que $V_1 \cap V_2 = \phi$, et E' = V_1 x V_2 (Rigo, 2010).
- **Étoile :** est un cas particulier de sous graphe bipartie ou X est un ensemble contenant le sommet central uniquement et Y contient le reste des nœuds (Koutra et al., 2015) .
- **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).

1.5 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 donnés ont été utilisés pour représenter un graphe, ces structures varient selon l'usage du graphe et la nature des traitements appliquer. Nous allons présenter dans cette partie les structures les plus utilisés.

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) v_1, v_2,..., v_m sont ordonnés de 1 à n et de 1 à m respectivement.

1.5.1 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, où les entrés (ij) prenent une valeur de "1" s'il existe un arc (une aréte dans le cas d'un graph non orienté) allant du sommet i au sommet j et un "0" sinon, e.i (Lehman et al., 2010) (SABLIK, 2018) (IUT, 2012) :

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

Dans le cas d'un graphe non orienté, la matrice est symétrique par rapport a la diagonale descendante de gauche à droite . e.i. $m_{ij} = m_{ji}$, 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) :

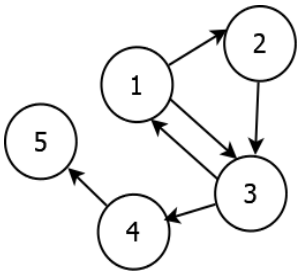

FIGURE 1.3 – Graphe orientée G

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

FIGURE 1.4 – Matrice d'adjacence du graphe G

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

1.5.2 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 ∈ V) et les colons représentent les arcs (j ∈ 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, la 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 de matrice d'incidence pour le graphe G ci-contre (figure 1.5) :

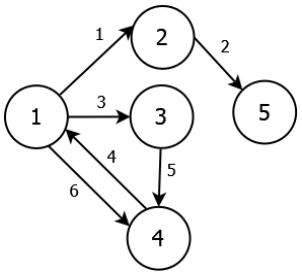

FIGURE 1.5 – Graphe orientée G

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

FIGURE 1.6 – Matrice d'incidence du graphe G

Place occupé en mémoire : $n \times m$

1.5.3 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 de matrice d'incidence pour le graphe G ci-contre (figure 1.7) :

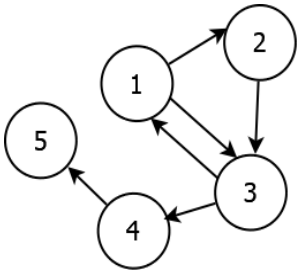

FIGURE 1.7 – Graphe orientée G

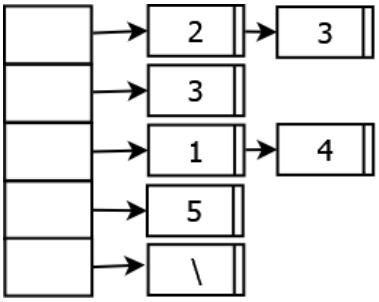

FIGURE 1.8 – Liste d'adjacence du graphe G

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.

1.6 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.

1.6.1 Graphes des réseaux sociaux :

Les réseaux sociaux représentent un lieu d'échange et de rencontre entre individus (entités) et dont l'utilisation est devenue de nos jours une nécessité. Pour représenter les interactions entre ces individus, nous avons généralement besoin de faire recours aux graphes où les sommets sont des individus ou des entités et les interactions entre eux sont représenté par des liens. Vue la diversité des interactions sociales, la modélisation de ces réseaux nécessite différents types de graphes : graphes non orientés pour pour les réseaux sociaux avec des relations non orientées, graphes orientés pour représenter des relations non symétriques comme c'est la cas dans les réseaux de confiance, graphes pondérés pour les réseaux sociaux qui contiennent différents niveaux d'intensités dans les relations, ... etc (Lemmouchi, 2012).

1.6.2 Graphes en Bioinformatique :

La bio-informatique est un domaine qui se trouve à l'intersection des deux grands domaines celui de l'informatique et celui de la biologie. Elle a pour but d'exploiter la puissance de 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 modélisation des réseaux d'interaction protéine-protéine (Pelle-

grini et al., 2004), le but dans ce cas est donc l'étude du fonctionnement des protéines par rapport à d'autre.

1.6.3 Le Graphe du web :

Le graphe du Web est un graphe orienté dont les sommets sont les pages du web et les arêtes modélise l'existence d'un lien hypertexte dans une page vers une autre (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.

1.7 Conclusion

Dans ce chapitre nous avons présenter les notions et les concepts généraux qui touchent a la théorie de graphes : définitions de graphes, leurs principales propriétés, leurs représentations et leurs domaines d'application.
Le point important qu'on a put tirer de cette partie est que les graphes sont devenue un moyen crucial et indispensable dans la modélisation des problèmes dans plusieurs domaines. Cependant ils devient de plus en plus complexe et volumineux suit a la grande quantités de données, ce qui rend leurs stockage, visualisation et traitement difficile. La compression de graphe est nait comme solution a ce problème. Dans le chapitre suivant nous allons présenter la compression de graphe, son rôle et ses différents méthodes.

Chapitre 2

Compression de graphe

2.1 Compression de données :

2.2 Compression appliquée aux graphes :

la compression de graphe a été proposé comme solutions pour le traitement et le stockage des graphe 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., 2018) :

— 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 tout types de données numérique accroissent d'une manière exponentiel, 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 graphe 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 central.

— 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 couteux 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 donnés 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 bruité, il 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 donnés. la compression permet donc de filtrer le bruits et de ne mettre en évidence que les donnés importantes.

2.2.1 Les types de compression :

2.2.2 Les métriques d'évaluation des algorithmes de compression :

2.3 Classification des méthodes de compression :

2.4 Méthodes de compression basés sur les k2-trees :

2.5 Méthodes de compression basés sur l'extraction de motifs :

2.5.1 VOG : Vocabulary Based Summarization of Graphs

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

La problématique traité par VoG peut être résumé comme suite : étant donné un graph statique non orienté G, on cherche a trouver un ensemble de sous-graphes chevauchés pour décrire de manière succinct le graphe donné en entré tout en minimisant le cout du codage en utilisant le principe MDL.

Le principe Minimum Description Length MDL est un concept de la théorie de l'information qui permet de codifier un ensemble de donnés en se basant sur un modèle, tout en minimisant le cout de codage par une fonction objective. Il peut être définit comme suite : étant donné un ensemble de modèles M, choisir celui qui minimise la fonction suivante : $min(D,M) = \mathrm{L}(M) + \mathrm{L}(D \mid M)$ où $\mathrm{L}(M)$ est la longueur, en bits de la description de M et $\mathrm{L}(D \mid M)$ est la longueur, en bits de la description des donnés en utilisant le modèle M. Pour utiliser le principe MDL dans VoG, il est d'abord essentiel de définir l'ensemble de modèles M, comment un modèle M peut il décrire un graphe et comment l'encoder. Nous notant aussi que pour une comparaison équitable entre les différents modèles, MDL nécessite que la description soit sans perte.

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

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

Pour calculer le cout totale de description général, on construit d'abord une approximation $\mathbf{M}$ de la matrice d'adjacence, en utilisant le modèle M : nous considérons de

manière itérative chaque structure $s \in M$ et complétons la connectivité de area(s) dans **M**. Comme on a $\mathbf{M} \neq A$ et comme notre méthode est une méthode sans perte, on aura besoin d'une matrice d'erreur E pour reconstituer la matrice d'adjacence tel que E= A $\bigoplus$ **M**.

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

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

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

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

Le codage des structures se fait selon leurs type :

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

$$\mathrm{L}(fc) = L_{\mathbb{N}}(|fc|) + \log \begin{pmatrix} n \\ |fc| \end{pmatrix}$$

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

$$\mathrm{L}(nc) = L_{\mathbb{N}}(|nc|) + \log \begin{pmatrix} n \\ |nc| \end{pmatrix} + \log(|\mathrm{area}(nc)|) + ||nc||l_1 + ||nc||'l_0$$

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

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

$$\mathrm{L}(fb) = L_{\mathbb{N}}(|A|) + L_{\mathbb{N}}(|B|) + \log \begin{pmatrix} n \\ |A| \end{pmatrix} + \log \begin{pmatrix} n - |A| \\ |B| \end{pmatrix}$$

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

$$\mathrm{L}(nb) = L_{\mathbb{N}}(|A|) + L_{\mathbb{N}}(|B|) + \log \begin{pmatrix} n \\ |A| \end{pmatrix} + \log \begin{pmatrix} n-|A| \\ |B| \end{pmatrix} + \log(|\mathrm{area}(nb)|) + ||nb||l_1 + ||nb||'l_0$$

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

$$\mathrm{L}(st) = L_{\mathbb{N}}(|st|\text{-}1) + \log \mathrm{n} + \log \begin{pmatrix} n-1 \\ |st|-1 \end{pmatrix}$$

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

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

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

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

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

1. Génération des sous-structures : Dans cette phase, Les méthodes de détection de communautés et de clustering sont utilisé pour décomposer le graphe en sous-graphes pas forcément disjoints. La méthode de décomposition utilisé dans VOG est SlashBurn.
2. Étiquetage des sous-graphes : L'algorithme cherche pour chaque sous-graphe généré dans l'étape précédente la structure $x \in \Omega$ qui le décrit le mieux, en tolérant un certain seuil d'erreur.
    a Étiquetage des structures parfaites : Tout d'abord, le sous-graphe est testé par rapport au structures complètes du vocabulaire (e.i. clique , étoile, chaine et noyau bipartie) pour une similarité sans erreur. Le test des cliques et des chaines est basé sur la distribution des degrés, Plus précisément, si tous les sommets d'un sous graphe d'ordre n ont un degré égale a n-1, il s'agit alors d'une clique. De même si tous les sommets ont un degré de 2 sauf deux sommets ayant le degré 1, le sous-graphe est une chaine. D'un autre coté, le sous-graphe est considéré comme noyau bipartie si les amplitudes de ses valeurs propres maximales et minimales sont égales, pour pouvoir identifier les sommets de chaque ensemble du noyau nous utilisons le parcours BFS avec la coloration des sommets. Quant a l'étoile, elle est considéré comme un cas particulier d'un noyau bipartie, il suffit donc que l'un des ensemble soit composé d'un seule sommet.
    b Étiquetage des structures approximative : Si le sous graphe ne correspond pas à une structures complète, on cherche la structure qui l'approxime mieux en terme de MDL. Pour ce faire, on le représente sous forme de chaque type $x \in \Omega$ et on choisit le type avec la description minimal.
    Remarque : Pour les structure parfaites (clique, étoile, chaine et noyau bipartie), en plus du cout d'encodage de la structure, on rajoute le cout de l'erreur local c'est a dire, $\mathrm{L}(x^*) + \mathrm{L}(E^+_{x^*}) + \mathrm{L}(E^-_{x^*})$ où $\mathrm{L}(x^*)$ est la description de la structure, $\mathrm{L}(E^+_{x^*})$ et $\mathrm{L}(E^-_{x^*})$ sont les arêtes incorrectement modélisés et les arêtes non modélisés respectivement dans area(x^*).

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

3. Assemblage du modèle : Dans cette dernier étape, une sélection d'un ensemble de structures parmi ceux de $\mathcal{C}$ est réaliser, des heuristiques de sélections sont utilisés 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 utilisés dans VOG on trouve :
   - PLAIN : Cette heuristique retourne tout les structures candidates. e.i. $M = \mathcal{C}$.
   - TOP-K : Cette heuristique sélectionne les k meilleurs candidats en fonction de leurs gain en bits.
   - GREEDY'N FORGET(GNF) : Parcours structure par structure dans l'ensemble $\mathcal{C}$ ordonnés par leurs qualité (gain en bits), ajoute la structure au modèle tant que elle n'augmente pas le cout total de la représentation, sinon l'ignore.

Ci-dessous le pseudo algorithme de VOG :

Algorithm 1 Pseudo Algorithme VOG

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

2.6 Méthodes de compression basés les k2-trees :

k^2-tree est une structure de donnés dense conçue à l'origine pour la compression des graphes du web, l'algorithme de base a été proposé par Bisaboa et all dans leur article k^2-trees for Compact Web Graph. (Brisaboa et al., 2009) Elle a été appliquer 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 donnés RDF (Alvarez-Garcia et al., 2017).

En général, l'algorithme peut être appliquer a 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 central. il offre aussi la possibilité de naviguer dans le graphe sans le décompresse, 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-air * de hauteur h = $[log_k$ n], chaque nœud contient un seul bit avec deux valeurs possible : 1 pour les nœuds interne 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. La racine correspond au premier niveau et prend une valeur 0. 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'= $k^{log_k n}$.

Pour construire l'arbre, k^2-trees commence par deviser la matrice en k^2 sous matrice 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 ajouté comme un fil 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 répéter de manière récursive sur les sous matrice représentés par des 1. k^2 sous matrices sont créé a chaque subdivision. L'opération est répéter jusqu'à ce que la subdivision atteins les cases de la matrice qui représenterons les feuilles de l'arbre au dernier niveau.

La figure 2.1 illustre la représentation k^2-trees d'une matrice de taille 10×10, étendue a une taille 16×16 pour un k=2 (Brisaboa et al., 2015).

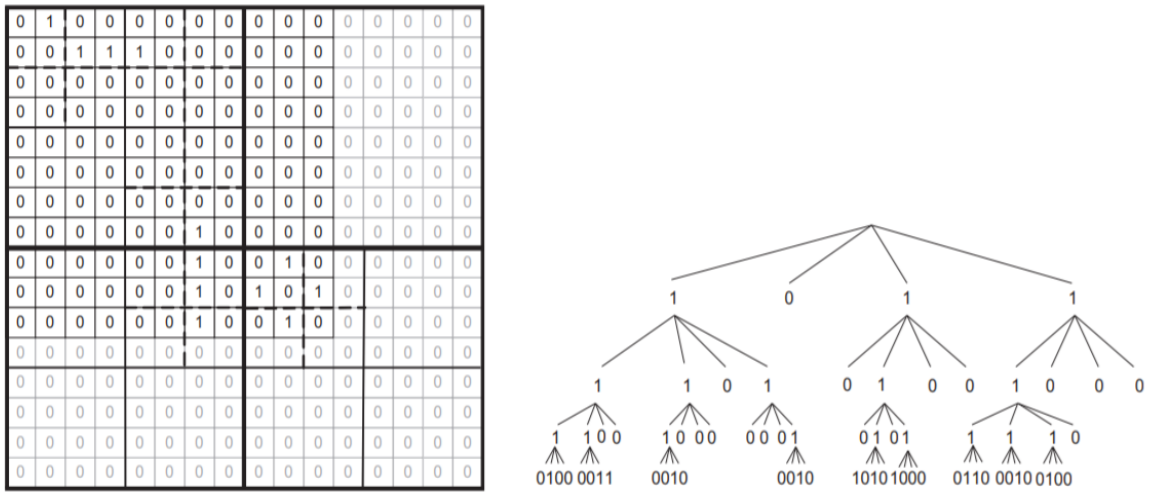

FIGURE 2.1 – Exemple de représentation k^2-trees d'une matrice d'adjacence d'un graphe

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 2.1) :

T = 1011 1101 0100 1000 1100 1000 0001 0101 1110

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^2m($log_{k^2}\frac{n^2}{m}+O(1)$), 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 ont proposé une hybridation qui consiste a changer la valeur du paramètre k en fonction du niveau de l'arbre en donnant a k une grande valeur au début pour réduire le nombre de niveaux et améliorer ainsi le temps de recherche, et une petite valeur a 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 proposés 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 travaux qu'on a put trouver.

Dans (Shi et al., 2012), les auteurs proposent deux techniques d'optimisation de l'algorithme : la première consiste à trouver un certain ordre de 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, et quand k est dynamique le problème est plus compliqué. Comme solution, les auteurs utilisent DFS avec des heuristiques pour trouver une approximation de l'ordre optimal. Cette optimisation permet de réduire le nombre de nœuds internes et produire ainsi un arbre optimal. 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 (Brisaboa et al., 2014b), les auteurs apportent deux amélioration principales 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, cella à plusieurs avantages, premièrement, l'espace est réduit étant donné que la taille de chaque arbre est en fonction de $\frac{n^2}{k^2}$,deuxièmement 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 occurrence dans L par

des pointeurs, cela permet d'éviter la redondance et réduire par suit la taille de la structure. Les pointeur sont représenté par des codes de longueur variable ordonné, 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) (Brisaboa et al., 2013) pour garantir un accès rapide au pointeur et conserver ainsi une navigation efficace.
Exemple : Pour la figure 2.1, le vocabulaire et L sont représentés comme suit :
V= [0010 0100 0011 1010 1000 0110]
L = $c_1c_2c_0c_0c_3c_4c_5c_0c_1$

dk^2-trees

Dans (Brisaboa et al., 2012), les auteurs développent la représentation k^2-trees pour les graphes dynamique. Ils proposent une nouvelle structure nommé 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 replacé 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 feuilles est une valeurs paramétré. Les noeuds interns des deux arbres permettent d'accéder au feuilles et de les modifier.
Chaque nœud interne de T_tree contient 3 éléments : deux compteurs b et o qui contiennent respectivement le nombre de bits et le nombre de uns stocké 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 2.2 présente une représentation dk^2-trees :

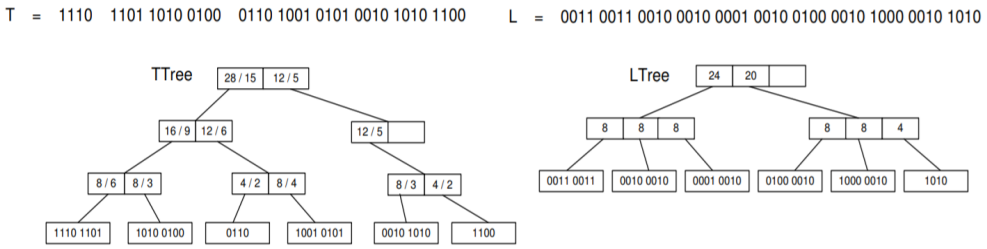

FIGURE 2.2 – Exemple d'une représentation dk^2-trees

k^n-trees

Dans (De Bernardo et al., 2013), Sandra and al présente k^n-trees, une généralisation des k^2-trees pour les problème multidimensionnelles. Cette méthodes a plusieurs applications, elle est utilisé pour représenter les bases de donnés multidimensionnels, les rasters et les graphes dynamique. k^n-trees repose sur k^2-trees pour représenter une matrice à n-dimensions, la matrice est décomposer en k^n sous-matrice 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-matrice sont induites, elles sont représenter par des nœuds dans l'arbre comme dans l'algorithme de base. Les structure utilisés pour le stockage sont aussi les même (T et L).

En posant n=3, la méthode peut être appliqué sur les graphe dynamique ou temporelles. Ce type de graphes est représenter par une grille à 3 dimension $X \times Y \times T$, où les deux premières dimensions représentent les nœuds de départ et de destination, et la troisième dimension représente le temps. Une telle représentation peut facilement être stocker avec k^3-trees. La figure 2.3 présente une représentation k^3-trees d'un graphe dynamique :

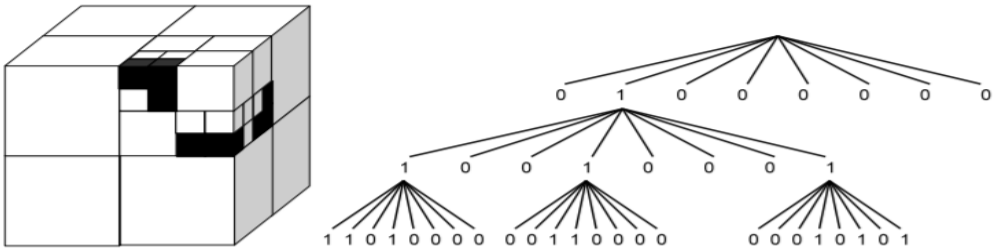

FIGURE 2.3 – Exemple d'une représentation k^3-trees

K^2-tress1

Une autre variante de la méthode a été proposé par (de Bernardo Roca, 2014). La représentation de base regroupe seulement les zones de zéros, puisque elle a été conçue au début pour les graphe du web qui possèdent une matrice d'adjacence extrêmement creuse. Les auteurs proposent d'étendre cette représentation en regroupant les zones de uns également. L'idée générale est d'arrêter la décomposition de la matrice d'adjacence quand une zone unis est trouvé, à savoir des zéros où des uns. Pour distinguer entre les différents nœuds, une représentation quadtree est utilisée (De Berg et al., 1997) : une couleur est attribué à chaque nœud, blanc pour une zone de zéro, noir pour une zone de uns et gris pour les nœuds internes e.i. les zones contenant des uns et des zéros. Pour le stockage des nœuds, les auteurs ont proposé quatre encodages présenter 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) où noires (1). Nous aurons donc : 10 pour les nœuds gris, 01 pour les nœuds noir et 00 pour les nœuds blancs. Notant que les feuilles du dernier niveau sont représenter par un seule bit. Après le codage, les premier bits de chaque nœud sauf ceux du dernier niveau sont stocker dans T, un autre tableau T' est créer pour sauvegarder les deuxièmes bits, les nœuds du dernier niveau sont stocker dans L.

k^2-$trees1^{2-bits}$: Le même principe de l'encodage précédant sauf que les nœuds gris

sont représenter par un seul bit, toujours à 1, le tableau T' va contenir dans ce cas la couleur des feuilles, cela 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 2 bits pour les nœuds noirs et gris, compte tenue de la fréquence des nœuds blanc dans les graphe du monde réel par rapport au autres. Nous aurons donc : 0 pour les noeuds blancs, 10 pour les nœuds gris et 11 pour les nœuds noir.

k^2-$trees1^{5-bits}$ **:** le dernier encodage repose sur la représentation de base, un nœud blanc est représenté par 0, un nœud noir où 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 ajouté au nœud noir pour le distinguer.

La figure 2.4 illustre une représentation k^2-tress1 avec les quatre encodage :

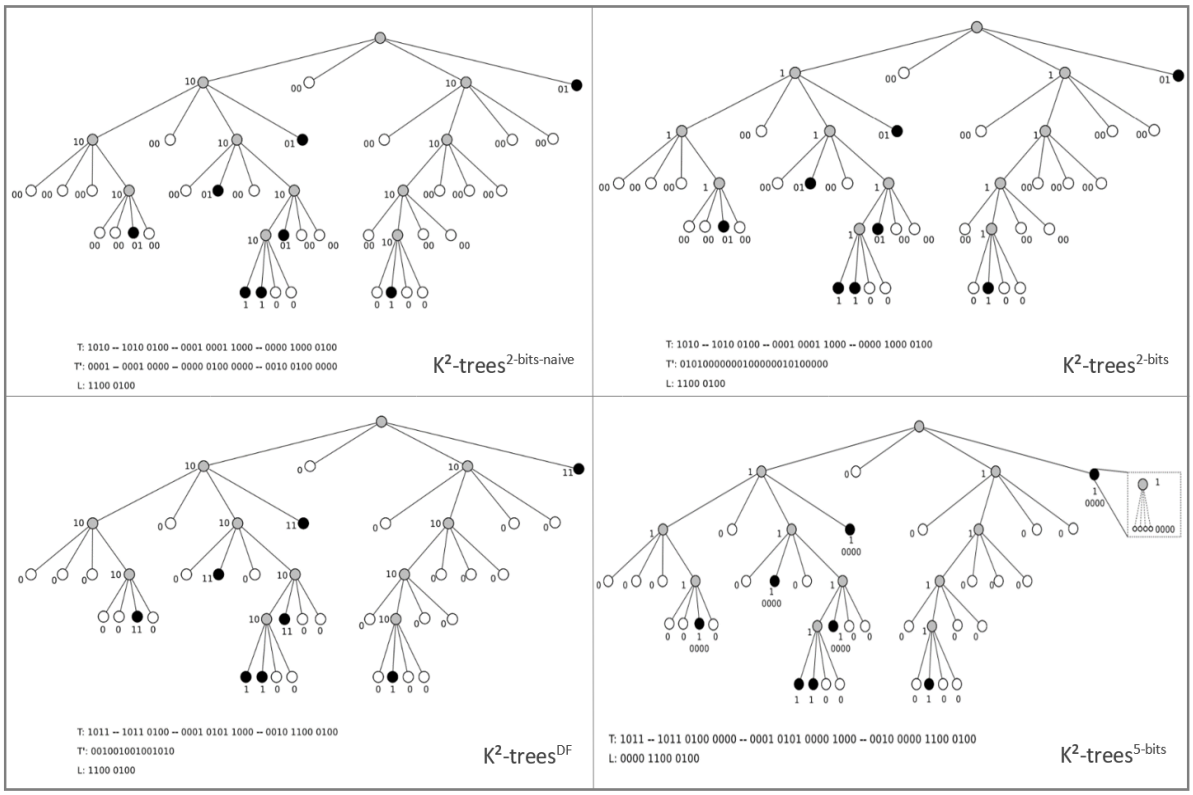

FIGURE 2.4 – Exemple de représentation k^2-trees1 d'une matrice d'adjacence d'un graphe

Delta-K^2-tress

Dans (Zhang et al., 2014), les auteur 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 appelé *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]. e.i. :

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

Où count1s compte le nombre de 1 dans une ligne, countDif compte le nombre de bits différent entre deux ligne et $\bigoplus$ représente le ou exclusif.
Pour déterminer si une ligne est identique a celle de la matrice d'adjacence, un tableau D est utilisé : Si D[i]=1 la ligne est identique ,sinon elle 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 certains lignes (lignes de différence), le graphe doit être décompresser et la matrice d'adjacence reconstituer

La figure 2.5 présente un exemple de la représentation Delta-k^2-trees

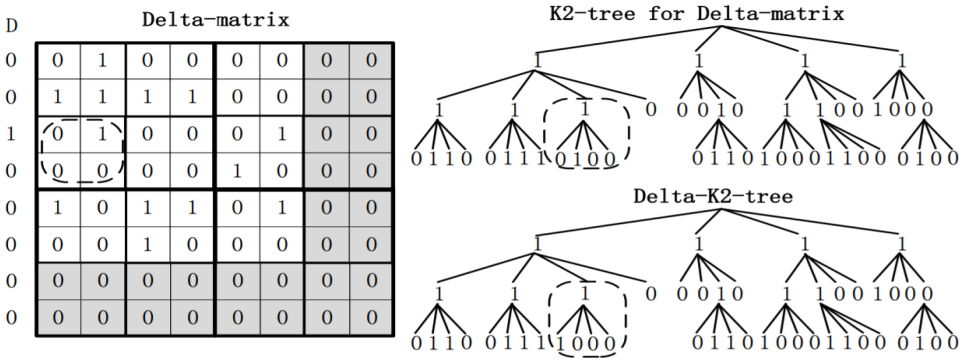

FIGURE 2.5 – Exemple de représentation Delta-k^2-trees d'une matrice d'adjacence d'un graphe

K^2-treaps

K^2-treaps est une autre variante de k^2-trees,elle a été proposé dans (Brisaboa et al., 2014a). Cette variante combine les k^2-trees avec une autre structure de donnés appelé treaps (Aragon and Seidel, 1989), les auteurs applique cette méthode sur des grilles multidimensionnels comme OLAP pour pouvoir les stocker et répondre efficacement aux requêtes top-K (Badr, 2013). La méthode peut être également appliqué sur les graphe pondéré, où chaque case de la matrice d'adjacence du graphe comporte le poids de l'arrêt qu'elle représente au lieu d'un 1. Une décomposition récursive en k^2 sous-matrice est appliqué sur la matrice d'adjacence et un arbre k^2-air est construit comme dans l'algorithme de base, comme suit : la racine de l'arbre va contenir les coordonnés de la cellule avec le plus grands poids de la matrice, ainsi que sa valeur. La cellule qui vient d'être ajouté à l'arbre est ensuite supprimé de la matrice. Si plusieurs cellules ont la même valeurs maximales, 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 avec ses coordonnés et sa valeurs, et supprimer la cellule choisis au final. La procédure contenue sur chaque branche de l'arbre jusqu'à ce qu'on tombe sur les cellules de la matrice d'origine où sur une sous matrice complètement vide.
La figure 2.6 suivante illustre le représentation k^2-treaps d'un graphe pondéré (Badr,

2013) :

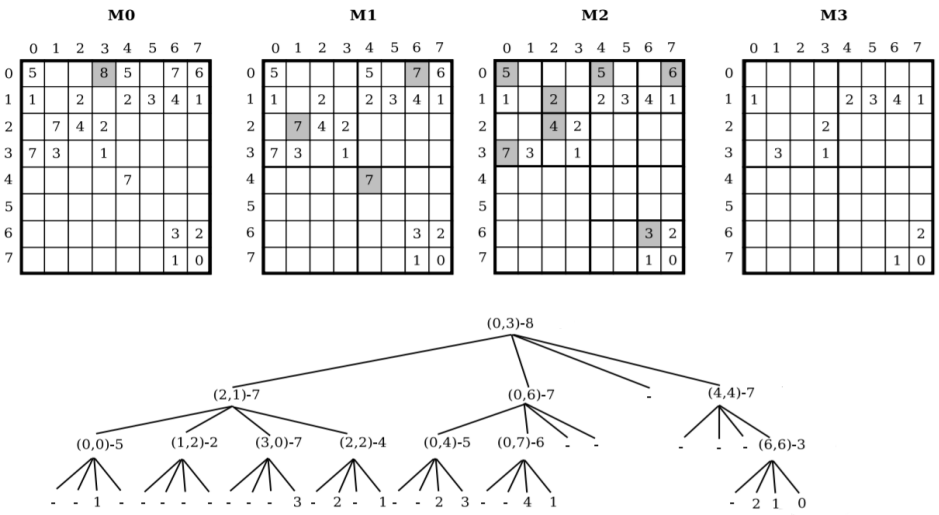

FIGURE 2.6 – Exemple de représentation k^2-treaps d'une matrice d'adjacence d'un graphe pondéré

Structure de donnés : Pour avoir une bonne compression, k^2-treaps effectue des transformation sur les donnés stockés . La première transformation consiste a changer les coordonné représenté dans l'arbre en des coordonné relative par rapport a 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 donnés sont utilisés pour sauvegarder les coordonnés et les valeurs des cellules ainsi que la topologie de l'arbre. Chaque structure est détaillé dans ce qui suit :

— *Listes de coordonnés locales :* la séquence de coordonnés de chaque niveau l de l'arbre est stocké dans une liste $coord[l]$.

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

— *L'arbre :* la structure de l'arbre k^2-treaps est sauvegarder avec un arbre k^2-trees, les

nœuds contenant des valeur dans k^2-treaps sont représenter par des uns, les nœuds vides par des zéros. Pour le stockage de l'arbre, un seul tableau T est utilisé.

La figure 2.7 représente les structures de donnés utilisés pour le stockage de l'arbre de la figure 2.6 précédente (Badr, 2013) :

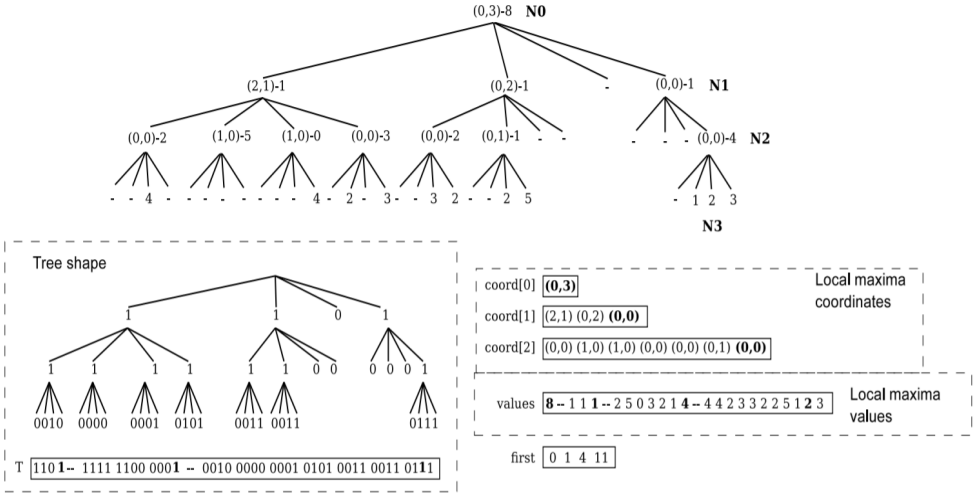

FIGURE 2.7 – Structures de donnés pour une représentation k^2-treaps d'une matrice

Ik^2-trees

Une autre représentation a été proposé par (Garcia et al., 2014), intitulé Ik^2-trees pour Interleaved k^2-trees, Elle est appliqué sur les bases de donnés RDF ainsi que sur les graphe dynamique. Les auteurs proposent cette méthodes pour les relations ternaires, Une relation ternaire est définit par un triplet T=$\{ x_i,y_j,t_k \} \subseteq X \times Y \times T$, I$k^2$-trees transforme cette relation en |Z| relation binaire . Dans les graphe dynamique, les deux premières dimension correspondent aux nœuds sources et destination et la troisième dimension reflète le temps. Le graphe est donc définit par |T| matrices d'adjacence prise à des instants t_k différents. Ik^2-trees représente les matrices simultanément, chaque matrice est représenté par un arbre k^2-trees, les arbre sont par la suit regroupé dans un seul arbre. Chaque nœuds de l'arbre obtenue 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 est donné par le nombre de uns de son parent. l'arbre final est stocké comme dans l'original k^2-trees avec deux tableau : T et L.

La figure 2.8 est un exemple de Ik^2-trees appliqué sur un graphe dynamique représenté sur trois instants

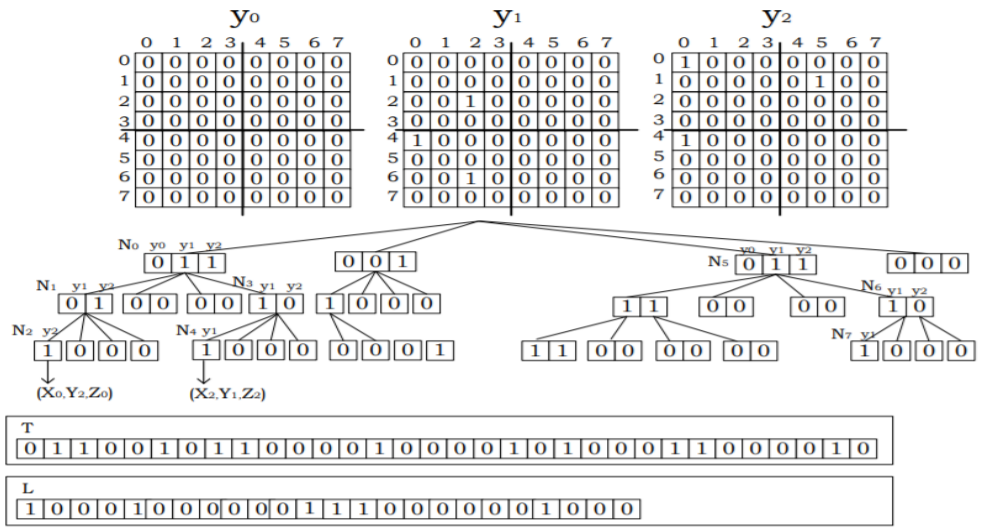

FIGURE 2.8 – Exemple d'une représentation Ik^2-trees

diff Ik^2-trees

Une variante de Ik^2-trees appelé Differential Ik^2 -tree a été étudié dans (Alvarez-Garcia et al., 2017), son but est d'améliorer le taux de compression en représentant uniquement les changements survenue 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 stocké. A l 'instant t_k, pour k>0, seuls les arrêtes qui change de valeurs entre t_{k-1} et t_k sont stockés. Les matrices sont représenter a 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 requêtes.

La figure 2.9 montre un exemple d'une représentation diff Ik^2-trees.

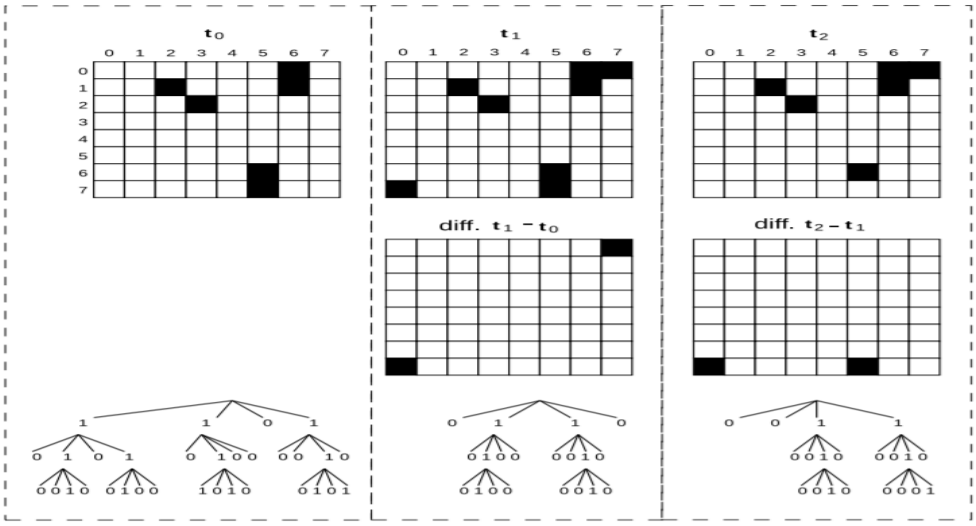

FIGURE 2.9 – Exemple d'une représentation diff Ik^2-trees

Att K^2-trees

Dans (Álvarez-García et al., 2018), les auteurs étendent la représentation k^2-trees pour les bases de donnés orienté graphe. Ces graphes sont étiquetés, attribués, orientés et ont des arrêtes multiples. Ils présentent le graphe sous forme d'une nouvelle structure intitulé Att k^2-trees pour Attributed k^2-trees.
La figure 2.10 montre un exemple de graphe pris en compte par Att K^2-trees

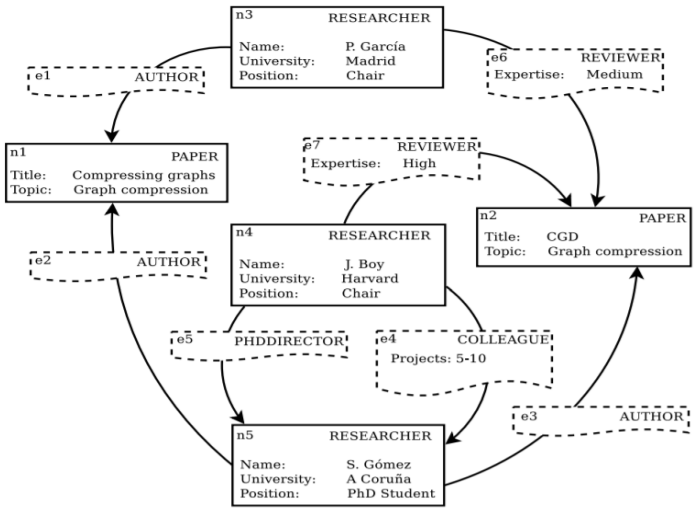

FIGURE 2.10 – Exemple d'un graphe étiqueté, attribué, orienté et multiple

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

— *Schéma :* Ce composant gère les étiquette 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 : représenter par un tableau qui contient les étiquètes des nœuds du graphe ordonné lexicographiquement. Un identifiant est attribué a chaque nœud du graphe selon l'ordre du tableau, les m_1 possédant la première étiqueté 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 stocké le plus grand identifiant portant son étiquette, cela permet de trouver l'étiquette d'un nœud a travers son identifiant.

Un schéma d'arrêtés : Comme dans le cas des nœuds, un tableau est utilisé

pour stocker les étiquette des arrêtes 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 arrêt, et d'accéder a ses attributs.

— *Donnés :* Ce composant contient tous les valeurs que peut prendre un attribut dans le graphe. Un attribut peut être représenter 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 a chaque apparition, ils sont stocker dans des listes et indexer avec l'identifiant de l'élément.

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

La figure 2.11 illustre les deux composants schéma et donnés de la représentation Attk^2-trees de la figure 2.10 précédente

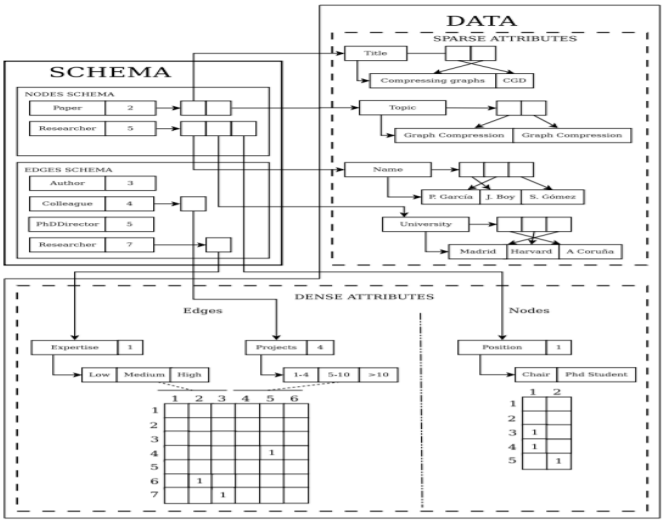

FIGURE 2.11 – Exemple d'un schéma et donnés de la représentation Attk^2-trees

— *Relations :* C'est le dernier composant de Attk^2-trees, il stocke les relations entre les nœuds et les arrêtes du graphe en utilisant un arbre k^2-trees et d'autres struc-

tures pour sauvegarder les identifiants des arrêts ainsi que les arrêtes multiples. Les structures supplémentaires sont les suivants :

Multi : Un tableau qui indique si l'arrête est multiple où non.

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

Next : Un tableau qui contient les identifient des arrêtes multiples restantes.

La figure 2.12 donne la représentation Attk^2-trees des relations du graphe de la figure 2.10 :

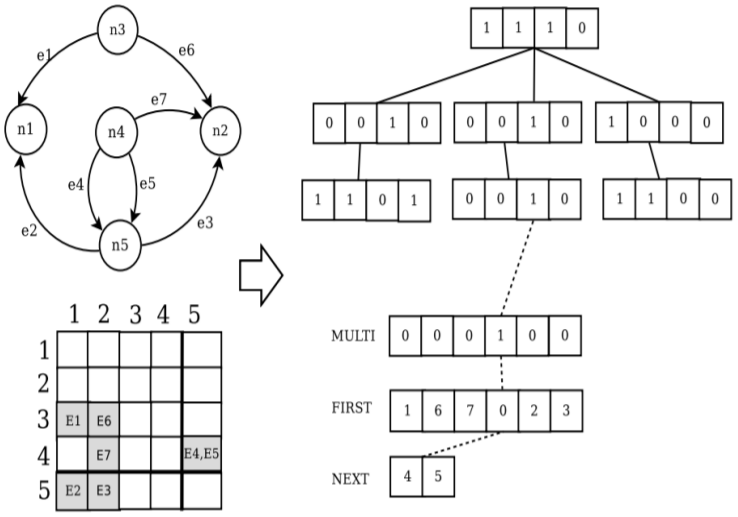

FIGURE 2.12 – représentation des relations dans Attk^2-trees

dynAttk^2-trees

Dans le même article (Álvarez-García et al., 2018), les auteurs étendent Attk^2-trees pour les graphes dynamiques, il proposent une nouvelle variante appelé 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, donnés et relations. Les composants sont semblables a ceux de Attk^2-trees mais avec certaines amélioration vue la nature dynamique du graphe.

Structure de donnés :

— *Schéma :* En ce qui concerne les nœuds, leurs étiquettes sont stockés dans une liste dynamique ordonné lexicographiquement. En outre, une séquence dynamique est utilisé pour sauvegarder le type de chaque nœud, elle est stocké ensuite sous forme d'un arbre d'ondelettes (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 sauvegardé 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 stockés les identifiants des arrêtes et les arrêtes multiples.

2.7 Conclusion

Chapitre 3

chapitre 03 : étude empirique

Definition 3.0.1. Here is a new definition

Deuxième partie

Conclusion

Random citation (Seo et al., 2018) embeddeed in text.

Random citation (Brisaboa et al., 2009) embeddeed in text.

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.

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.

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.

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.

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

Koutra, D., Kang, U., Vreeken, J., and Faloutsos, C. (2015). Summarizing and understanding large graphs. *Statistical Analysis and Data Mining : The ASA Data Science Journal*, 8(3) :183–202.

Lehman, E., Leighton, F. T., and Meyer, A. R. (2010). Mathematics for computer science. Technical report, Technical report, 2006. Lecture notes.

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. (2018). Graph summarization methods and applications : A survey. *ACM Computing Surveys (CSUR)*, 51(3) :62.

Lopez, P. (2003). Cours de graphes.

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

Parlebas, P. (1972). Centralité et compacité d'un graphe. *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.

Roux, P. (2014). *Théorie des graphes*.

SABLIK, M. (2018). Graphe et langage.

Seo, H., Park, K., Han, Y., Kim, H., Umair, M., Khan, K. U., and Lee, Y.-K. (2018). An effective graph summarization and compression technique for a large-scaled graph. *The Journal of Supercomputing*, pages 1–15.

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.

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