Thèse de doctorat / octobre 2019

Université Panthéon - Assas

Ecole doctorale d'économie, gestion, information et communication

Thèse de doctorat en informatique

soutenue le 24 octobre 2019

Approximation dynamique de clusters dans un graphe social : Méthodes et Applications.

UNIVERSITÉ PARIS II
PANTHÉON-ASSAS

Guillaume VIMONT

Sous la direction de Michel DE ROUGEMONT

Rapporteur :

M. PRIEUR Christophe, Professeur associé à Télécom ParisTech
M. SALAMATIAN Kavé, Professeur à Polytech Annecy-Chambéry

Membres du jury :
Mme. CHEVALIER Céline, Maître de conférences à l'Université Paris II
M. SPYRATOS Nicolas, Professeur émérite à l'Université Paris-Sud

Avertissement

L'université n'entend donner aucune approbation ni improbation aux opinions émises dans cette thèse ; ces opinions doivent être considérées comme propres à leur auteur.

Remerciements

Je tiens à remercier et exprimer ma gratitude à mon directeur de thèse, le professeur Michel de ROUGEMONT qui a rendu ce travail possible. Ses précieux conseils, avis et suggestions ont été d'une valeur inestimable à toutes les étapes du travail.

Je tiens également à remercier mes collègues, camarades du Ministère des Armées, pour tout le soutient qu'ils m'ont apporté lors de la réalisation de cette thèse.

Enfin, je voudrais remercier ma famille pour tout leur amour et leurs encouragements. Pour mes parents qui m'ont soutenu dans toutes mes activités, et mon amie qui a été solidaire, encourageante et patiente, au cours de cette thèse.

Je vous remercie.

Guillaume VIMONT
Université Panthéon-Assas
Juin 2019

Résumé :

Nous étudions comment détecter des clusters dans un graphe défini par un flux d'arêtes, sans stocker l'ensemble du graphe. Nous montrons comment détecter de gros clusters de l'ordre de $\sqrt{n}$ dans des graphes qui ont $m = O(n \log(n))$ arêtes, tout en stockant $\sqrt{n}.\log(n)$ arêtes. Les graphes sociaux suivent le régime où m satisfait cette condition. Nous étendons notre approche aux graphes dynamiques définis par les arêtes les plus récentes du flux et à plusieurs flux. Nous proposons des méthodes simples et robustes afin de détecter ces clusters de manière approchée.

Nous définissons la corrélation de contenu de deux flux $\rho(t)$ par la similarité de Jaccard de leurs clusters, dans les fenêtres au temps t. Nous proposons une méthode simple et efficace pour approcher cette corrélation en ligne et montrons que pour les graphes aléatoires dynamiques qui suivent une loi de puissance, nous pouvons garantir une bonne approximation.

Une des applications est l'analyse des flux Twitter. Nous calculons les corrélations de contenu de ces flux en ligne. Nous proposons ensuite une recherche par corrélation où les réponses aux ensembles de mots-clés sont entièrement basées sur les petites corrélations des flux. Les réponses sont ordonnées par les corrélations, et les explications peuvent être tracées avec les clusters stockés.

Descripteurs : Algorithmes de streaming, graphes dynamiques, Clustering, Approximation

Abstract :

We study how to detect clusters in a graph defined by a stream of edges, without storing the entire graph. We show how to detect large clusters in the order of $\sqrt{n}$ in graphs that have $m = O(n \log(n))$ edges, while storing $\sqrt{n}.\log(n)$ edges. Social graphs satisfy this condition (m). We extend our approach to dynamic graphs defined by the most recent stream of edges and multiple streams. We propose simple and robust methods based on the approximation to detect these clusters.

We define the content correlation of two streams $\rho(t)$ is the Jaccard similarity of their clusters in the windows before time t. We propose a simple and efficient method to approach this online correlation and show that for dynamic random graphs that follow a power law, we can guarantee a good approximation.

As an applications we follow Twitter streams and compute their content correlations online. We then propose a search by correlation where answers to sets of keywords are entirely based on the small correlations of the streams. Answers are ordered by the correlations, and explanations can be traced with the stored clusters.

Keywords : Streaming algorithms, Dynamic graphs, Clustering, Approximation

Sommaire

Introduction

Avec l'apparition de plateformes Web comme Facebook et Twitter, les réseaux sociaux font aujourd'hui partie de notre vie quotidienne. Bien que les réseaux sociaux et leurs analyses soient un domaine de recherche très populaire depuis des décennies en sociologie [60, 21, 22], la récente révolution de l'Internet et des applications informatiques a rendu disponible une énorme quantité de données du monde réel à analyser et à traiter pour les chercheurs, sous forme de flux de données.

Les données issues des réseaux sociaux sont très différentes des données traditionnelles utilisées en exploration de données. Outre leur immense taille, les données principalement générées par les utilisateurs sont bruitées et non structurées, avec d'abondantes relations sociales telles que des amitiés et des suiveurs. Ce nouveau type de données exige de nouvelles approches d'analyses computationnelles des données, qui peuvent combiner des théories sociales avec des méthodes statistiques d'exploration de données [6, 14, 50, 44].

Dans cette thèse, nous nous intéresserons aux graphes sociaux, où les arêtes sont présentées sous forme de flux de données et nous allons introduire des algorithmes approchés qui permettent d'approximer leurs clusters, sans stocker tout le graphe. Ce problème de la détection de cluster est un problème fondamental [3, 63, 11, 19, 18], en particulier dans le livre de Easley & Kleinberg [22] qui lie l'analyse de données à la valeur économique résultante.

En 2015, les travaux [12, 32, 46, 27] de ces quatre articles se concentrent sur les solutions algorithmiques pour trouver des sous-graphes denses, à partir de la borne inférieure linéaire $O(n)$ en mémoire, de l'article [8]. Un des problèmes récents est de détecter avec une mémoire sous-linéaire des clusters dans des flux de données, en utilisant de nouvelles hypothèses.

Le résultat principal de cette thèse est la description d'algorithmes de streaming en mémoire sous-linéaire. Nous pouvons ensuite considérer plusieurs flux simultanés et définir l'intégration de données dans ce cadre. Les résultats principaux sont :

— Nous utilisons une hypothèse sur les graphes sociaux : la distribution de degrés suit une loi de puissance. De manière plus précise, nous allons montrer qu'avec une mémoire de $O(\sqrt{n}.\log(n))$, il est possible de trouver des communautés de taille $O(\sqrt{n})$.

— Nous définissons une notion de corrélation entre deux flux d'arêtes et une distance entre flux qui est ensuite étendue à une distance entre tags. Ensuite, nous présentons un moteur de recherche basé sur ces distances et donc sur ces corrélations.

Contexte

Les réseaux sociaux génèrent des flux de données, plus précisément des flux d'arêtes d'un graphe. Ces graphes ont des propriétés structurelles et statistiques. Comment utiliser ces propriétés pour trouver des algorithmes efficaces sur ces flux de données ?

Dans un graphe aléatoire Erdös-Renyi [28], la distribution des degrés des sommets suit une loi Gaussienne. Cependant, les distributions de degrés des graphes sociaux ne sont pas homogènes mais suivent une loi de puissance. Les arêtes peuvent être plus denses au sein de certains groupes de sommets [34, 37] et créent ainsi un cluster. Cette caractéristique des graphes sociaux est générale et nous pouvons voir un cluster ou une communauté comme un grand sous-graphe dense. La détection et l'estimation de ces clusters sont des sujets importants.

Nous nous intéressons à l'approximation de clusters dans un flux d'arêtes, utilisons des techniques d'échantillonnage dynamique d'un graphe et montrons comment suivre certaines évolutions structurelles à partir des échantillons. Les algorithmes approchés sont basés sur l'analyse des composantes connexes des échantillons. L'échantillonnage permet de réduire le bruit présent dans les données, et ces méthodes sont moins sensibles au bruit des données.

L'échantillonnage dynamique nous permet d'apporter des solutions face aux trois problèmes dominants lors de l'exploitation de données issues des réseaux sociaux, qui sont la taille, le bruit et le dynamisme. Nous allons nous intéresser, en particulier, à des clusters de taille suffisamment grande de l'ordre de $O(\sqrt{n})$, dans des graphes [1] qui ont $m = O(n \log(n))$ arêtes. Si on a un cluster de taille $O(\sqrt{n})$, alors nous pouvons garantir avec notre algorithme basé sur l'échantillonnage à partir d'un réservoir, la détection du cluster. Dans le cas où le cluster est de taille inférieure à $O(\sqrt{n})$, nous constatons en pratique que l'algorithme détecte également le cluster, mais nous ne pouvons pas le garantir.

Nous allons montrer que l'algorithme qui teste si la taille de la composante géante du réservoir est supérieure à un certain seuil, elle va nous fournir une bonne approximation, pour la détection d'un cluster dans un graphe qui suit une distribution de degrés, selon une loi de puissance.

1. Si la distribution des degrés d'un graphe suit une loi de puissance, alors le nombre d'arêtes $m = O(n \log(n))$

Contributions

Nous avons étudié la manière de détecter des clusters dans un graphe social défini par un flux d'arêtes, sans stocker l'ensemble du graphe, en utilisant une mémoire de $O(\sqrt{n}.\log(n))$. Les résultats principaux de la détection de clusters sont :

— Si le graphe a un γ-cluster [2] de taille supérieure à $O(\sqrt{n})$, nous allons pouvoir détecter l'existence de ce cluster avec grande probabilité à l'aide d'un échantillonnage par réservoir. Nous pouvons également l'approximer en considérant le $2-$*core* de la composante géante du réservoir. Inversement, si le graphe est un graphe aléatoire qui suit cette loi statistique, nous allons démontrer avec une grande probabilité qu'il n'y a pas de cluster. Ces résultats sont détaillés dans le chapitre 4.

— Nous avons étendu cette technique au graphe dynamique défini par une fenêtre temporelle. Si nous avons un graphe dynamique dans lequel apparaît un gros cluster pendant un certain temps, nous allons le détecter avec une grande probabilité. Inversement, s'il n'y a pas de grand cluster, nous allons pouvoir détecter avec une grande probabilité son inexistence. Ces résultats sont détaillés dans le chapitre 5.

La détection de grands clusters par les algorithmes précédents permet d'envisager *l'intégration de plusieurs flux de données*. La fusion de données est un sujet lié à la théorie du signal qui utilise aussi des notions d'approximation. L'intégration de données est un concept utilisé dans les Bases de Données, lorsque plusieurs sources de données coexistent, mais il n'y a pas, en général, d'approximation. Nous utilisons l'approximation des clusters pour définir la *corrélation de flux* et le principe de la *recherche par corrélation*.

— Nous définissons la corrélation de contenu $\rho(t)$ de deux flux, comme la similarité de Jaccard de l'union de leurs clusters, dans les fenêtres au temps t. Nous proposons une méthode simple et efficace pour approcher cette corrélation en ligne et montrons que pour les graphes aléatoires dynamiques qui suivent une loi de puissance, nous pouvons garantir une bonne approximation.

— Grâce à l'analyse des corrélations des flux, nous pouvons associer des distances entre flux et construire un arbre phylogénique qui reproduit ces distances. Nous définissons une distance entre deux tags en étendant les distances entre flux et construisons un moteur de recherche basé sur ces distances. Les résultats principaux de l'intégration de flux sont présentés dans le chapitre 6.

Une des applications des résultats de cette thèse, est l'analyse de flux Twitter. Nous calculons les corrélations de contenu de ces flux en ligne. Nous proposons ensuite une recherche par corrélation à partir de mots-clés, où les réponses aux ensembles de mots-clés sont entièrement basées sur les petites corrélations des flux. Les réponses sont ordonnées

2. Un γ-cluster est un sous-graphe dense où le nombre d'arêtes internes est défini par $|E(S)| \geq \gamma.|S|.\frac{|S|-1}{2}$

par les corrélations, et les explications peuvent être tracées avec les clusters stockés.

Comment l'analyse de données permet de créer de nouveaux services et de nouvelles valeurs économiques ? Deux sujets ne sont pas abordés dans le livre de Easley & Kleinberg [22] : les algorithmes de flux et l'intégration de données. Ce sont précisément ces deux sujets qui sont les contributions de cette thèse. Les algorithmes de flux sont approchés au sens probabiliste et permettent de définir une intégration approchée de données. Cette thèse est basée sur les publications suivantes :

— The content correlation of multiple streaming edges, *IEEE International Conference on Big Data*, 2018" [25].

— Approximate Integration of streaming data, *Entrepôts de Données et l'Analyse en Ligne, Business Intelligence & Big Data*, 2017" [24]

— Approximate Integration of streaming data, (version étendue de l'article 2017, dans la série Springer *Information search, integration, and personalization* 2019 [26]).

— The value of analytical queries on Social Networks, *Workshop Mining Big Data in Social Networks, IEEE International Conference on Big Data*, 2015" [23].

Organisation de la thèse

Dans le chapitre 1 nous introduisons les concepts de théorie des graphes et en particulier les différents modèles de graphes aléatoires. Nous précisons les différentes notions de clusters.

Dans le chapitre 2, nous présentons le réseau social Twitter. Nous allons présenter plusieurs méthodes pour récupérer les données depuis ce réseau social, puis nous allons transformer ces données en un flux d'arêtes et analyser ses propriétés, une fois converties en graphe.

Dans le chapitre 3, nous présentons les méthodes classiques de détection de clusters dans un graphe connu.

Dans le chapitre 4, nous nous intéresserons aux méthodes d'approximation de clusters dans un graphe social. Nous présentons des méthodes d'échantillonnage basées sur des réservoirs statiques. Il s'agit d'une approche qui permet d'approximer des clusters dans un graphe.

Dans le chapitre 5, nous étudions la détection de clusters dans des graphes sociaux dynamiques. Nous présentons des méthodes d'approximation de clusters au sein de ces graphes, ainsi que des méthodes d'échantillonnage basées sur des réservoirs dynamiques. Parmi ces méthodes d'échantillonnage nous présentons notre algorithme, le "step reservoir sampling".

Dans le chapitre 6, nous étendons l'approximation de clusters issus des réseaux sociaux à l'intégration de flux ; dans un premier temps, nous définissons la corrélation entre différents flux de graphes dynamiques et définissons une distance entre tags. Dans un second temps, nous présentons un moteur de recherche basé sur ces distances et donc sur les corrélations de ces flux de graphes dynamiques.

1 Graphes et réseaux sociaux

L'analyse de réseaux sociaux utilise plusieurs concepts de la théorie des graphes dans ses algorithmes et ses métriques. La théorie des graphes nous fournit des outils pour représenter formellement les réseaux sociaux et quantifier les propriétés structurelles du réseau. Les définitions qui sont présentées dans cette section suivent largement celles de Biggs [13] et de Bollobás [15].

1.1 Théorie des graphes

Un réseau est défini comme un objet composé d'entités interagissant les unes avec les autres via certaines connexions. Le moyen naturel de représenter mathématiquement un réseau passe par un modèle de graphe.

1.1.1 Définitions

Un graphe G est constitué d'un ensemble de nœuds V et d'un ensemble d'arêtes E. L'ensemble E est constitué de paires de nœuds (u_1, u_2) où u_1, $u_2 \in V$. Les arêtes peuvent être dirigées ou non-dirigées si le graphe est symétrique.

Un sous-graphe G_s peut être généré à partir de G soit en sélectionnant un sous-ensemble V_s de V, soit par un sous-ensemble E_s de E. Un sous-graphe G_s généré à partir d'un sous-ensemble V_s présente toutes les arêtes de G qui connectent des nœuds dans V_s.Un sous-graphe G_s généré à partir d'un sous-ensemble E_s de E, présente les nœuds de G qui existent dans E_s.

L'analyse de réseaux sociaux utilise principalement les sous-graphes générés par les nœuds. Deux nœuds qui ont une arête en commun sont adjacents, de même deux arêtes qui ont un nœud commun sont adjacents. Pour les graphes non-orientés, un nœud qui se trouve dans la paire d'arête est incidente par rapport à l'arête et l'arête est incidente par rapport au nœud. Pour les graphes orientés, une arête e de u_1 à u_2 est incidente à u_2 et est incidente à partir de u_1.

Degré. Le degré (ou valence) d'un nœud u est le nombre d'arêtes qui lui sont incidentes. Étant donné un graphe symétrique $G = (V, E)$ la somme des degrés d'un nœud u est donnée par l'équation suivante :

$$\sum_{u \in V} d(u) = 2\,|E| \tag{1.1}$$

Dans un graphe orienté, nous pouvons distinguer le degré d'un nœud u par son degré entrant $d^-(u)$ défini comme le nombre d'arêtes dirigées vers le nœud u et le degré sortant $d^+(u)$ défini comme le nombre d'arêtes sortants du nœud u.

Densité. La densité d'un graphe $G = (V, E)$ est définie par le rapport entre le nombre d'arêtes divisé par le nombre d'arêtes possibles. La densité d'un graphe non-orienté est donc égale à :

$$D = \frac{2\,|E|}{|V| \cdot (|V| - 1)} \tag{1.2}$$

Pour un graphe orienté, la densité est égale à :

$$D = \frac{|E|}{|V| * (|V| - 1)} \tag{1.3}$$

Notons que le nombre maximal d'arêtes est :

$$\frac{|V| * (|V| - 1)}{2} \tag{1.4}$$

Un autre concept utile est le degré moyen ou la connectivité d'un graphe défini comme :

$$\frac{2\,|E|}{|V|}. \tag{1.5}$$

1.1.2 Distribution des degrés

Dans un graphe non-orienté $G = (V, E)$, la distribution des degrés est définie par :

$$P(k) = \frac{|\{u_i \in V : deg\, u_i = k\}|}{|E|}, \tag{1.6}$$

$P(k)$ donne la fraction (ou le pourcentage) des nœuds dans V, dont le degré est $k \in \{0, 1, ..., n-1\}$, $n = |G|$. Par conséquent, $P(k)$ peut être interprété comme la probabilité pour un nœud (tiré au hasard de V) d'avoir exactement k arêtes.

Dans les graphes aléatoires Erdös-Rényi, il a été montré [6] que $P(k)$ suit une distribution de Poisson dont le pic est situé à $\langle k \rangle$ ($\langle . \rangle$ désigne la valeur d'attente).

Cependant, dans de nombreux réseaux sociaux, $P(k)$ suit une distribution de loi de puissance, à savoir :

$$P(k) \sim k^{-\lambda} \tag{1.7}$$

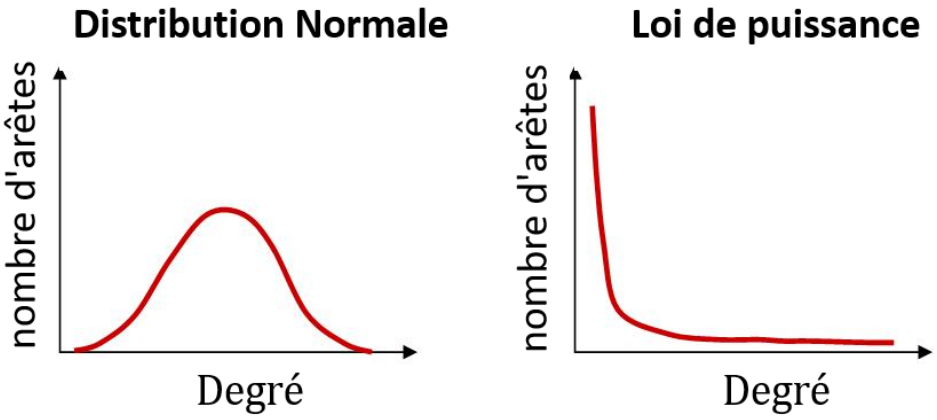

FIGURE 1.1: Distribution de puissance

1.1.3 Les graphes aléatoires

Un graphe aléatoire est un graphe généré par un processus aléatoire, celui-ci peut être obtenu de différentes manières. Une des possibilités est de choisir un graphe parmi une collection de graphes possibles avec une certaine probabilité, une autre possibilité consiste à choisir une arête avec une certaine probabilité.

1.1.3.1 Le graphe aléatoire uniforme

Definition 1. *Un graphe aléatoire tiré selon $G(n, m)$ est un graphe obtenu en échantillonnant uniformément tous les graphes non-isomorphes à n sommets et m arêtes.*

La probabilité de sélectionner un graphe $\widehat{G}$ nécessite de déterminer la taille de l'ensemble de tous les graphes possibles. Elle est calculée en choisissant uniformément un sous-ensemble de m arrêtes, parmi les $n(n–1)/2$ arêtes possibles. La probabilité de $\widehat{G}$ est donnée par :

$$P(G) = \frac{1}{\binom{\frac{n}{2}}{m}} \tag{1.8}$$

1.1.3.2 Le modèle Erdös-Renyi

Nous pouvons également générer un graphe aléatoire en décidant pour chaque arête si on la garde avec une probabilité p.

Definition 2. *Un graphe aléatoire $\widehat{G}$ tiré selon $G(n, p)$ est obtenu en partant de l'ensemble des sommets $V = 1, 2, 3...n$, et en reliant chaque paire de sommets i, j où $i \neq j$ par une arête avec la probabilité p tel que $0 \leq p \leq 1$.*

Ce modèle est généralement appelé le modèle de graphe aléatoire Erdös-Renyi (ER), décrit par Erdös-Renyi dans deux documents de 1959 [29] et de 1960 [30].

1.1.3.3 Le modèle de configuration

Les graphes de type Erdös-Renyi ont une distribution de degrés de type gaussienne, or dans les graphes issus des réseaux sociaux nous n'observons pas ce type de distribution. Nous introduisons donc le configuration model qui est un modèle de graphe aléatoire plus général et qui intègre n'importe quelle distribution de degrés.

Une distribution de degré arbitraire $\mathcal{D}$, est une séquence qui détermine le nombre de nœuds de degré 1, de degré 2, de degré i. Étant donnée cette séquence, on a le nombre de nœuds et le nombre d'arêtes qui sont la somme des degrés divisés par deux.

Definition 3. *Étant donné une séquence de degrés arbitraire $\mathcal{D}$, un graphe aléatoire $\widehat{G}$ tiré selon le configuration model ($\mathcal{D}$) est un graphe aléatoire à n sommets, où en fonction de leurs degrés on associe d_i demi-arêtes à chaque nœuds u_i selon $\mathcal{D}$. On choisit un appareillement uniforme entre les demi-arêtes ; les demi-arêtes sont choisies symétriquement de manière aléatoire entre-elles, afin de former des arêtes.*

La propriété fondamentale du modèle de configuration est la probabilité d'avoir une arête entre les sommets i et j :

$$p_{ij} = \frac{d_i d_j}{2m} \tag{1.9}$$

Par exemple, pour une séquence de degrés $\mathcal{D} = (3, 2, 1)$, nous pouvons générer un graphe aléatoire à partir de cette séquence. En faisant un matching aléatoire uniforme sur les "stubs" (demi-arêtes) attachés aux sommets, nous pouvons en déduire que chaque matching de ce type se produit avec une probabilité égale. Toutefois, cela ne signifie pas que chaque graphe se produira avec une probabilité égale, car certains matching produisent le même graphe. La figure 1.2 illustre le processus.

Remarque (Problème de réalisation de graphe)**.** *Le problème consiste à savoir s'il existe un graphe qui a $\mathcal{D}$ comme séquence de degrés. La condition de Erdös-Gallai permet de tester une telle existence [31]. La condition de Erdös-Gallai stipule qu'il existe un graphe à partir d'une séquence de degrés $d_1 \geq ... \geq d_n \geq 0$, si et seulement si ces deux conditions sont vraies :*

- $d_1 + ... + d_n$ *est égal*
- $\forall 1 \leq k \leq n, d_1 + ... + d_k \leq k(k-1) + min(k, d_{k+1}) + .. + min(k, d_n)$

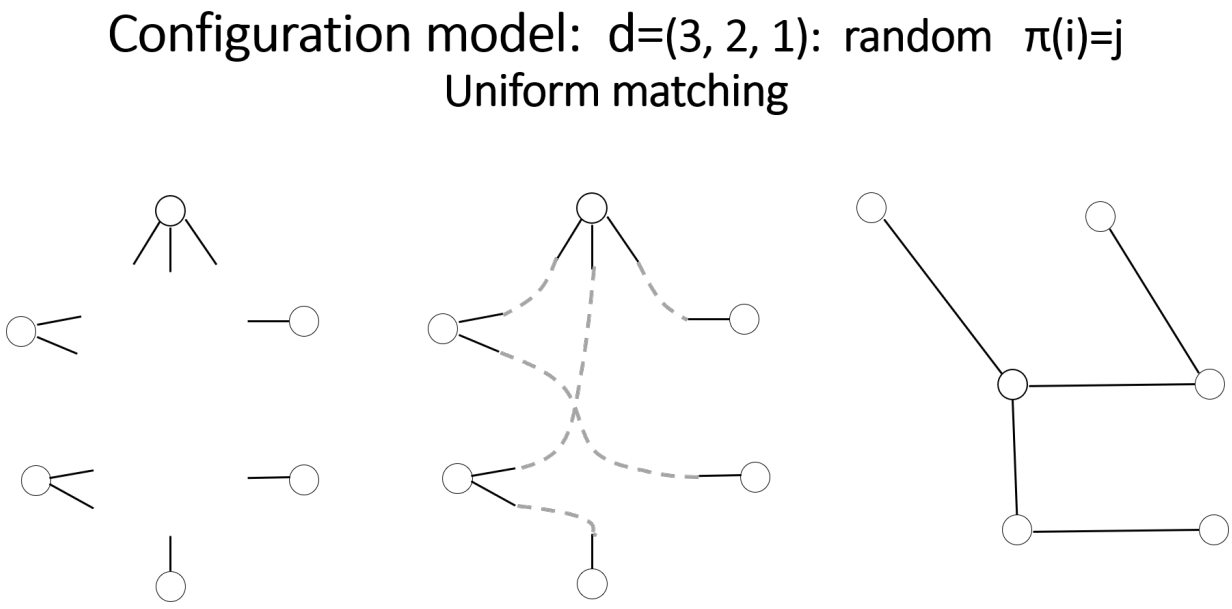

FIGURE 1.2: Exemple du modèle de configuration

1.1.3.4 Modèle d'attachement préférentiel

Le modèle d'attachement préférentiel est motivé par l'observation des distributions de la loi de puissance dans divers contextes.

Definition 4. *Un graphe aléatoire tiré selon $PA(n, m)$ est un graphe construit dynamiquement : étant donné $\widehat{G}_n$ au stade n, on construit $\widehat{G}_{n+1}$ en ajoutant un nouveau nœud et m arêtes reliant le nouveau nœud avec un nœud j suivant la distribution du degré dans $\widehat{G}_n$.*

Les graphes résultants ont une distribution des degrés qui suit une loi de puissance, telle que la loi de Zipf où :

$$Prob[d(i) = j] = \frac{c}{j^2} \tag{1.10}$$

Le modèle Barabasi-Albert [10] est un processus permettant de générer des graphes aléatoires à l'aide du modèle d'attachement préférentiel, ce processus peut être vu comme un algorithme.

La probabilité p_i que le nouveau nœud soit connecté à un nœud i est proportionnelle au nombre d'arête que ces nœuds existants ont déjà :

$$p_i = \frac{d_i}{\sum_j d_j} \tag{1.11}$$

où d_i est le degré du nœud i, et la somme est faite sur tous les nœuds existants j.

Ce modèle possède une propriété intéressante qui fait que les « riches deviennent plus riches » ; en d'autres termes, il est plus probable que les nouveaux nœuds se connectent aux nœuds qui sont déjà fortement liés. Cela semble vrai pour les réseaux sociaux dans le sens où plus une personne a de connaissances, plus la probabilité qu'elle rencontre davantage d'individus au fil du temps augmente également. Ces types de distributions sont connus sous le nom de distributions de degrés de loi de puissance, vu dans la section précédente.

1.1.4 Composante géante

Une composante géante est une composante connexe d'un graphe aléatoire donné, qui contient une fraction constante des sommets du graphe.

Avec un graphe aléatoire $G(n, p)$, si la valeur de p est faible, alors on a une faible densité d'arêtes, et le graphe ne contient pas de composante géante. Une valeur plus élevée de p implique une densité d'arête élevée, et nous observons ainsi l'émergence de composantes géantes dans le graphe.

Le résultat fondamental d'Erdös-Renyi est de réaliser que si $p > \frac{1}{n}$ alors on a une composante géante et que la transition est brutale, c'est-à-dire que l'on a une transition de phase.

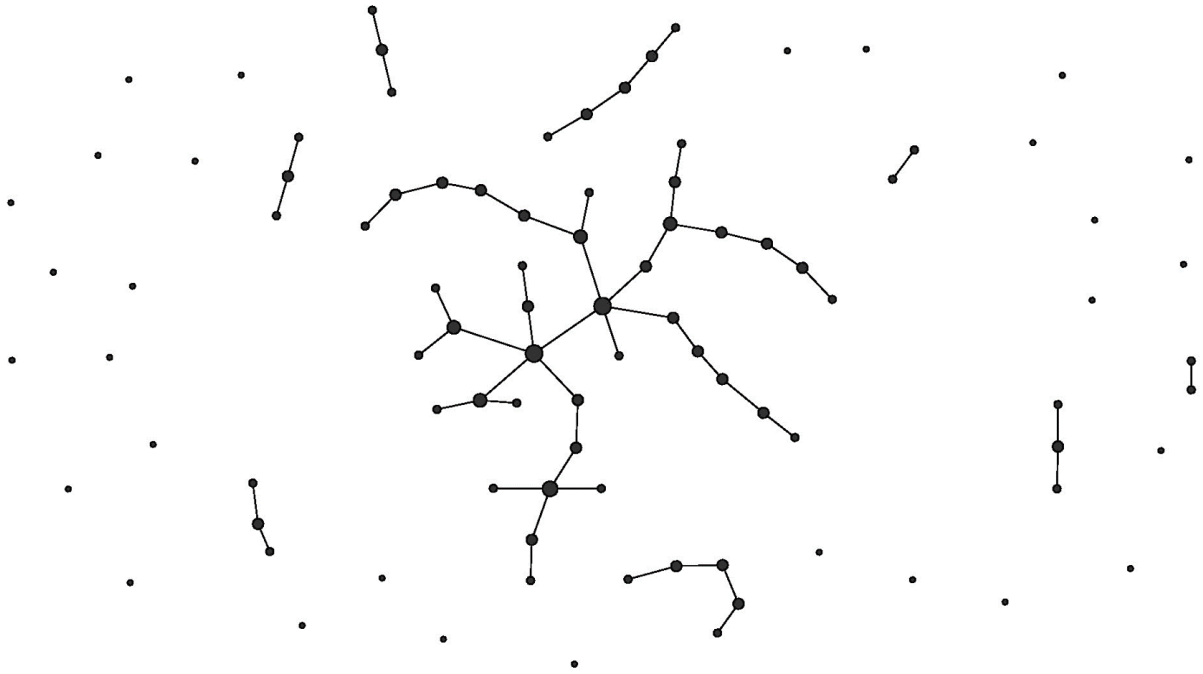

FIGURE 1.3: Exemple de Composante géante

Le moment on l'on peut détecter une composante géante fait partie de l'une des questions que l'on se pose dans cette thèse. Dans le configuration model, nous observons une composante géante. Un résultat fondamental des travaux de Molloy-Reed [49] expose que si $\mathbb{E}[\mathcal{D}^2] - 2\mathbb{E}[\mathcal{D}] > 0$ alors nous avons une composante géante.

De nombreuses études dans le monde de la théorie des graphes tournent autour de la composante géante ; ces recherches visent à trouver le moment où l'on a une composante géante, en particulier lorsque nous avons une distribution arbitraire de degrés $\mathcal{D}$. Dans notre cas, nous avons un réservoir d'arêtes et nous souhaitons savoir si nous avons des composantes géantes à l'intérieur.

1.2 Expérience de Milgram

Dans les années 1960, Stanley Milgram développe une expérience qui permet de découvrir le phénomène du « petit monde » [47]. Cette expérience est née d'un désir d'en apprendre davantage sur la probabilité que deux personnes choisies au hasard se connaissent [58].

Milgram a mené son expérience en demandant à différentes personnes de différentes villes d'envoyer une lettre à des cibles spécifiquement nommées qui n'étaient pas connues des expéditeurs. La condition imposée, les participants pouvaient seulement passer les lettres, de main à main, à des connaissances personnelles qu'ils pensaient être capables d'atteindre l'objectif, directement ou via les amis des amis.

Les lettres incluaient des informations expliquant le but de l'expérience et des informations de base sur le contact ciblé. En observant les résultats des différentes séries de lettres qui ont atteint la cible, Milgram a conclu que le degré moyen de séparation était de six.

En 1998, Duncan J. Watts et Steven H.Strogatz, ont publié le premier modèle de graphe concernant le phénomène du petit monde [61], c'est un graphe aléatoire obtenu par reconnexion des liens dans un cercle, au sein duquel seuls les voisins sont initialement connectés . Les réseaux Watts-Strogatz possèdent des propriétés liées au phénomène du petit monde car la probabilité de reconnexion β est suffisamment grande.

1.3 Clusters

> "La définition la plus simple d'un cluster est celle d'une composante connectée, et la définition la plus stricte est que chaque cluster doit être une clique maximale. Dans la plupart des cas, les clusters se situent quelque part entre ces deux extrêmes." (Satu Elisa Schaeffer [57])

Dans cette section, nous nous intéressons à la définition de cluster, il est important de souligner ici qu'il n'existe pas de définition unique de clusters dans un graphe. Les travaux de Satu Elisa Schaeffer [57] montre qu'il existe une multitude de définitions possibles.

La formulation pour définir un cluster dépend soit du domaine d'application, soit généralement du type de clusters que nous souhaitons mettre en valeur. Néanmoins, quelle que soit la définition, l'objectif reste identique : rassembler les noeuds de composantes denses.

Nous pouvons regrouper ces définitions en deux grandes familles, d'une part nous avons les définitions de clusters basées sur la densité du graphe, d'autre part nous avons les définitions de clusters basées sur la conductance dans le graphe.

1.3.1 Cluster basé sur la densité

Nous utiliserons dans cette thèse la définition ci-dessous :

Definition 5. *Soit $\gamma \leq 1$ et $E(S)$ un ensemble d'arêtes internes, c'est-à-dire les arêtes $e = (u, v)$ lorsque $u, v \in S$. Un γ-cluster est un sous-ensemble S tel que :*

$$|E(S)| \geq \gamma . |S| . \frac{|S| - 1}{2} \tag{1.12}$$

Il existe également d'autres définitions proches de cette définition pour définir un cluster, nous présentons celle du (α, β)-cluster :

Definition 6. *Étant donné un graphe $G = (V, E)$, où chaque sommet a une boucle, un sous-ensemble $C \subset V$ est un (α, β)-cluster si :*

1. *Densité interne : $\forall v \in C, |E(v, C)| \geq \beta |C|$*
2. *Extérieurement clairsemé : $\forall u \in V \setminus C, |E(u, C)| \leq \alpha |C|$*

Remarque. *Ces deux définitions partagent la première condition, cependant la deuxième condition n'est pas commune car nous ne souhaitons pas de coupe.*
Nous allons suivre la définition 5, car elle nous permet d'avoir la maîtrise de la transition de phase dans un sous-graphe dense.

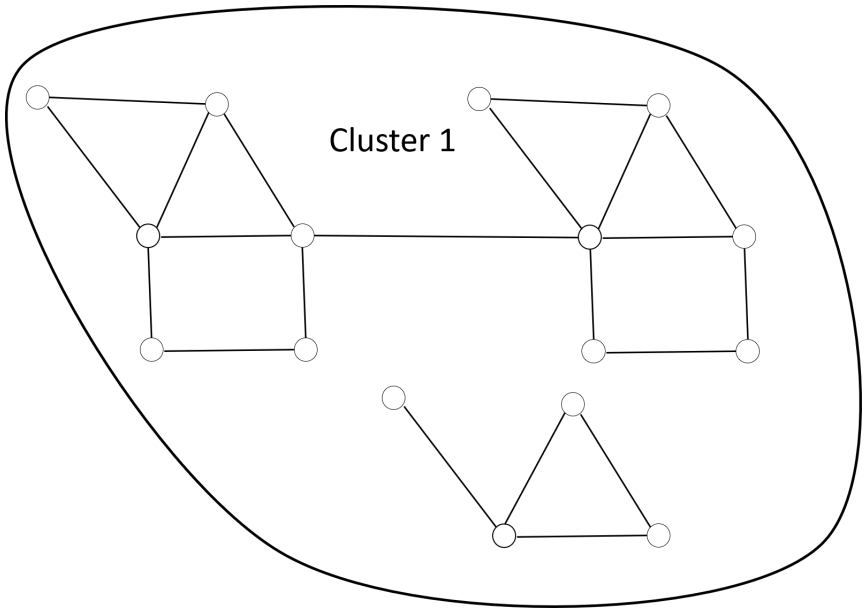

FIGURE 1.4: Exemple de détection de clusters dans un graphe G avec une approche basée sur la densité

Dans [46, 32, 12], on essaie de maximiser la densité. On considère la densité d'un sous-graphe comme le rapport $\frac{|E(S)|}{|S|}$ que l'on cherche à maximiser avec ρ^* définit comme :

$$\rho^* = max_S\{\frac{|E(S)|}{|S|}\} \tag{1.13}$$

Un sous graphe est dense si sa densité est proche de ρ^*.

1.3.2 Cluster basé sur la conductance

Une mesure naturelle de la connectivité d'un graphe, est sa conductance. La conductance d'une coupe [41] est une mesure qui compare la taille d'une coupe (c'est-à-dire le nombre d'arêtes coupées) et le poids des arêtes dans l'un ou l'autre des deux sous-graphes induits par cette coupe.

La conductance $\phi(G)$ d'un graphe est la valeur minimale de conductance entre tous ses clusters. Considérons une partition qui divise G en k groupes $C_1, C_2...C_k$ sans qu'ils ne se chevauchent. La conductance d'un groupe donné $\phi(C_i)$ [7] peut être obtenue comme il est indiqué dans l'équation 1.14, où $a(C_i) = \sum_{u \in C_i} \sum_{v \in V} w(u, v)$ est la somme du poids de toutes les arêtes avec au moins une extrémité dans C_i. Cette valeur $\phi(C_i)$ représente le coût d'une coupe qui divise G en deux ensembles de sommets C_i et $V \setminus C_i$.

$$\phi(C_i) = \frac{\sum_{u \in C_i} \sum_{v \notin C_i} w(\{u, v\})}{min(a(C_i), a(\bar{C_i}))} \tag{1.14}$$

$$\phi(G) = min(\phi(C_i)), C_i \subseteq V \tag{1.15}$$

Sur cette base, il est possible de définir le concept de conductance intra-cluster $\alpha(C)$ (voir équation 1.16) et la conductance inter-cluster $\sigma(C)$ (voir équation 1.17) pour un cluster donné $C = C_1, C_2, ...C_k$.

$$\alpha(C) = min_{i \in \{1,....,k\}} \phi(G[C_i]) \tag{1.16}$$

$$\sigma(C) = 1 - max_{i \in \{1,....,k\}} \phi(C_i) \tag{1.17}$$

Un bon cluster doit avoir des valeurs élevées de conductance intra- et inter-cluster.

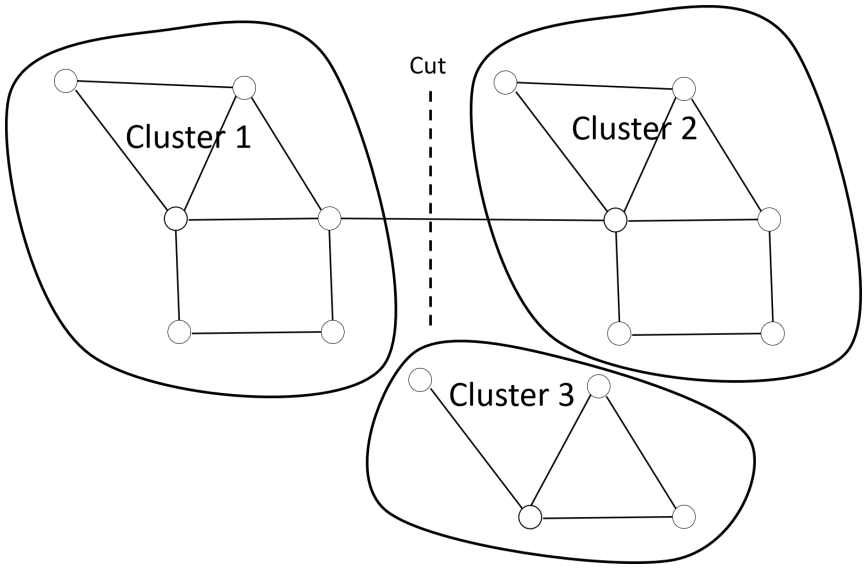

FIGURE 1.5: Exemple de détection de clusters dans un graphe G avec une approche basée sur la conductance

Nous présentons ci-dessous un (α, ϵ)-cluster, cette définition est présentée dans [41]. Pour mesurer la qualité d'un cluster, il faut utiliser deux critères, le premier étant la qualité minimale des clusters (appelé α), et le second, la fraction du poids total des arêtes qui ne sont pas couvertes par les clusters (appelée ϵ).

Definition 7. *Nous appelons une partition* $\{C_1, C_2, ..., C_i\}$ *de* V *est un* (α, ϵ) *-cluster si :*

1. *La conductance de chaque* C_i *est d'au moins* α*.*
2. *Le poids total des arêtes inter-clusters est au maximum une fraction de* ϵ *du poids total des arêtes.*

On obtient ainsi une mesure basée sur les deux critères de la qualité d'un clustering.

1.4 Coefficient de clustering

Ce coefficient mesure la tendance d'un graphe à former des groupes de nœuds fortement liés (c'est-à-dire connectés) et peut être défini au niveau local et au niveau global.
Pour un nœud particulier (niveau local), son coefficient de regroupement représente le degré de connexion entre ses voisins. Le voisinage N_i d'un nœud u_i donné dans un graphe est défini comme suit :

$$N_i = \{u_j : e_{ij} \in E \vee e_{ji} \in E\}\,. \tag{1.18}$$

Le coefficient de clustering C_i d'un nœud u_i est alors défini comme le rapport entre le nombre de liaisons entre les voisins de u_i et le nombre maximum de liaisons possibles. Pour un graphe orienté, le coefficient de clustering est donné par :

$$C_i = \frac{|\{e_{jk} : u_j, u_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)} \tag{1.19}$$

Pour un graphe non-orienté, le coefficient de clustering est donné par :

$$C_i = \frac{2\,|\{e_{jk} : u_j, u_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)} \tag{1.20}$$

Notons que $C_i \in [0, 1]$.

Au niveau global, le coefficient de clustering C du réseau entier est calculé comme la valeur moyenne de tous les coefficients de regroupements locaux :

$$C = \frac{1}{n}\sum_{i=1}^{n} C_i \tag{1.21}$$

Dans les graphes aléatoires qui sont censés être non-orientés, le coefficient de clustering dépend de la probabilité de connexion de deux nœuds aléatoires et peut être calculé à partir du degré moyen $\langle k \rangle$ comme :

$$C_{rand} = \frac{\langle k \rangle}{n} \tag{1.22}$$

Selon l'équation ci-dessus, les graphes aléatoires devraient avoir un faible coefficient de clustering, car le degré moyen $\langle k \rangle$ est généralement faible par rapport au nombre de nœuds du réseau, n. Les réseaux sociaux ont un coefficient de clustering beaucoup plus grand que ce que l'on attend d'un graphe aléatoire de même taille n. Cela est dû au fait que les réseaux du monde réel ont une structure sous-jacente considérablement différente de celle des réseaux aléatoires.

1.5 Distance et similarité

Les mesures de distance jouent un rôle très important pour mesurer la similarité entre les objets de données. La principale exigence du calcul de la métrique dans un problème spécifique consiste à obtenir une fonction de distance / similarité appropriée. Une fonction métrique ou une fonction de distance est une fonction qui définit une distance entre les éléments / objets d'un ensemble. Un ensemble avec une métrique est appelé espace métrique.

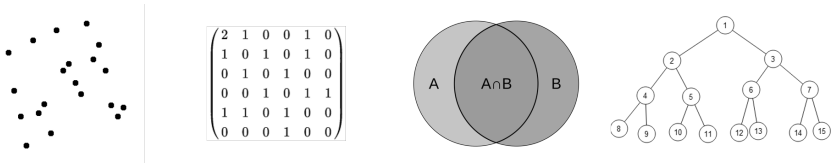

FIGURE 1.6: Exemple de représentation de données

Il existe deux principales classes de mesure de la distance : Euclidienne et non Non-euclidienne. Les distances euclidiennes sont conformes à notre concept physique de distance. Mais il existe de nombreuses autres mesures de distance qui peuvent être définies entre des échantillons à plusieurs variables.

Ces distances non-euclidiennes sont de types différents : certaines satisfont toujours une métrique, tandis que d'autres ne sont pas des métriques mais sont tout de même très utiles pour mesurer la différence entre les échantillons. Nous considérerons plusieurs mesures de distance non-euclidiennes qui sont populaires, telles que la distance L1 (également appelée distance de Manhattan), la distance des cosinus, la distance de Minkowski et l'indice de Jaccard.

Une mesure de similarité peut être définie comme étant la distance entre différents points de données. Alors que la similarité est une quantité qui reflète la force de la relation entre deux éléments de données, la dissemblance concerne la mesure de la divergence entre deux éléments de données [55]. La performance de nombreux algorithmes dépend de la sélection d'une bonne fonction de distance sur le jeu de données d'entrée[55].

Dans les sous-sections suivantes, vous trouverez un bref aperçu des fonctions de mesure de la similarité couramment utilisées dans la littérature :

Distance de Jaccard. L'indice de Jaccard mesure la similarité de deux éléments de données comme l'intersection divisée par l'union des éléments de données. La distance de Jaccard mesure la dissimilarité entre les ensembles. Elle consiste simplement à soustraire

l'indice de Jaccard à 1.

$$J_\delta(A,B) = 1 - J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|} \tag{1.23}$$

Distance Euclidienne. La distance euclidienne est considérée comme la métrique standard pour les problèmes géométriques. C'est simplement la distance ordinaire entre deux points. La distance euclidienne est largement utilisée dans les problèmes de clustering. La mesure de distance par défaut utilisée avec l'algorithme K-means est la distance euclidienne. La distance euclidienne détermine la racine carrée des différences entre les coordonnées d'une paire d'objets, comme cela est indiqué dans l'équation.

$$Dist_{X,Y} = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2} \tag{1.24}$$

Distance des cosinus. La mesure de distance des cosinus détermine le cosinus de l'angle entre deux vecteurs donnés. Ici, θ donne l'angle entre deux vecteurs et A, B sont des vecteurs à n dimensions.

$$cos\theta = \frac{A.B}{\|A\| \, \|B\|} \tag{1.25}$$

2 Le réseau social Twitter

Twitter est un réseau social de type microblogging créé en 2006, dans lequel les utilisateurs publient des messages appelés *tweets*. Il a été conçu à l'origine comme un service basé sur le SMS où les messages sont limités à 160 caractères. Ainsi, les *tweets* sont limités à 140 caractères, laissant 20 caractères pour le nom de l'utilisateur.

Il existe différents types d'interactions dans Twitter, les utilisateurs de la plateforme peuvent :

— suivre des utilisateurs (*following*) et être suivi (*followers*)

— interagir avec les *tweets* publiés à travers des actions telles que : répondre, j'aime, relayer, message privé.

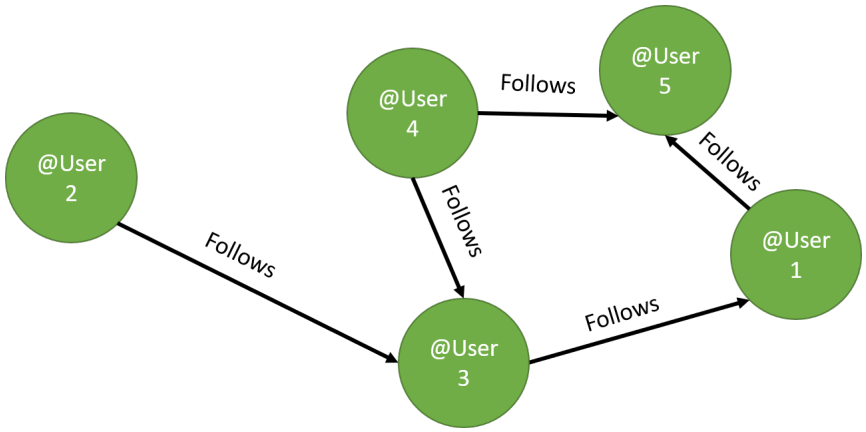

FIGURE 2.1: Exemple de graphe non-symétrique d'utilisateurs sur Twitter

Les utilisateurs de Twitter ont adopté différentes conventions telles que les mentions, les retweets et les hashtags dans leurs *tweets*. Les mentions, symbolisées par un @, indiquent que le tweet est une mention à un utilisateur. Les retweets sont utilisés pour republier le contenu d'un autre tweet, il est symbolisé par l'acronyme *RT* en début d'un *tweet*. Les hashtags sont utilisés pour désigner des mots-clés dans le message en préfixant un mot avec le symbole dièse, par exemple #France, #élections.

Les restrictions de taille et les mécanismes de partage de contenu de Twitter ont créé un dialecte unique [39] qui comprend de nombreuses abréviations, acronymes, mots mal orthographiés et émoticônes qui ne sont pas habituels dans les médias traditionnels. Les mots et les expressions fréquemment utilisés pendant une période donnée sont appelés « trending topics ». Ces « trending topics » sont répertoriés par la plateforme pour différentes régions du monde et peuvent également être personnalisés pour l'utilisateur.

Twitter est devenu la plateforme de microblogging la plus populaire, avec des centaines de millions d'utilisateurs diffusant quotidiennement des millions de messages personnels. Le volume important et abondant de données qui y sont diffusées fait de Twitter une source de données très intéressante, car c'est la plateforme qui partage le plus facilement ses données grâce à des API que nous allons présenter dans la section suivante.

2.1 Récupération des données

Les tweets publiés par des comptes publics peuvent être récupérés à l'aide de l'une des deux API Twitter suivantes :

1. API Search REST : Renvoie une collection de tweets correspondant à une requête spécifiée.
2. API Streaming : Renvoie un flux de tweets correspondant à un filtre spécifié.

Pour utiliser les API Twitter, il est nécessaire d'avoir un compte développé [1]. Une fois réalisé, il nous faut créer une application pour pouvoir obtenir des clés et jetons afin de s'authentifier sur ces API.

En effet, Twitter utilise le protocole Oauth pour déléguer les autorisations d'accès. Ce protocole permet d'autoriser une application (dite "consommateur") à utiliser l'API d'une autre application (dite "fournisseur") dans notre cas Twitter pour le compte d'un utilisateur.

1. https ://apps.twitter.com/

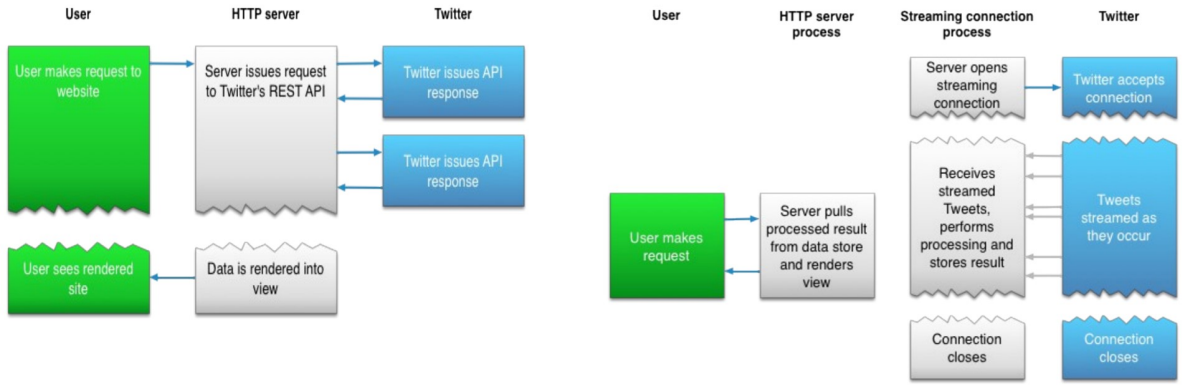

FIGURE 2.2

2.1.1 API Search Twitter

L'API search permet de retourner des tweets qui répondent à une requête q. Cette requête q peut être composée à partir de plusieurs paramètres tels que :

Paramètres	Obligatoire	Description	Example
q	oui	Renvoie les tweets concernant des termes contenus dans le message du tweet.	France
geocode	non	Renvoie les tweets d'utilisateurs situés dans un rayon donné de la latitude / longitude donnée.	37.781157 -122.398720 1mi
lang	non	Renvoie les tweets écrits dans une langue spécifique	fr
result_type	non	Renvoie le type de tweets par exemple les plus populaires (les plus retweetés), les plus récents ou mixtes (mélange entre les plus populaires et les plus récents).	mixed
count	non	Nombre de tweets retournés	100
until	non	Retourne les tweets créés avant une date donnée	2019-01-10
since_id	non	Renvoie les résultats avec un identifiant supérieur (c'est-à-dire plus récent) à l'identifiant spécifié.	12345
max_id	non	Retourne les résultats avec un ID inférieur à (ou antérieur à) ou égal à l'ID spécifié.	54321
include_entities	non	Les entités ne seront pas incluses si elles sont définies sur false.	false

Par exemple, si nous voulons recevoir les tweets les plus récents contenant le terme "france" écrit en français, nous devons réaliser la requête suivante :

```
$curl --request GET
--url 'https://api.twitter.com/1.1/search/tweets.json?q=france$lang=fr&result_type=recent'
--header 'authorization: OAuth oauth_consumer_key="consumer-key-for-app",
oauth_nonce="generated-nonce", oauth_signature="generated-signature",
oauth_signature_method="HMAC-SHA1", oauth_timestamp="generated-timestamp",
oauth_token="access-token-for-authed-user", oauth_version="1.0"'
```

Nous présentons la structure et le format de la réponse à la requête q dans la section suivante.

L'API search présente des limites :

1. Le nombre de tweets retournés par requête ne peut pas dépasser 1500 tweets.
2. Il ne peut pas trouver les tweets qui ont été envoyés il y a plus d'une semaine.

Ci-dessous nous présentons un exemple d'utilisation en Python [2] de l'API search de Twitter en utilisant la librairie Tweepy [3]

```
import tweepy


# consumer keys and access tokens, used for OAuth
consumer_key = 'YOUR-CONSUMER-KEY'
consumer_secret = 'YOUR-CONSUMER-SECRET'
access_token = 'YOUR-ACCESS-TOKEN'
access_token_secret = 'YOUR-ACCESS-TOKEN-SECRET'

# OAuth process, using the keys and tokens
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

# creation of the actual interface, using authentication
api = tweepy.API(auth)


# search recent tweets containing text "France"
# search french language tweets
# limit response 5 tweets
search = tweepy.Cursor(api.search, q="France", result_type="recent", lang="fr").items(5)

# print the searched tweets
for item in search:
    print (item.text)
```

Code 2.1: Exemple d'utilisation de l'API search en Python

2.1.2 API streaming Twitter

Les requêtes de flux s'exécutent en continu. Les différentes API de flux de données fournies par Twitter permettent aux développeurs d'avoir accès aux données de flux globales de Twitter. Twitter propose plusieurs points d'accès au flux de données :

1. Flux publics : Ils donnent accès aux flux de données accessibles au public sur Twitter. Généralement, ils sont utilisés pour suivre des sujets spécifiques, individuels et pour l'exploration de données.
2. Flux d'utilisateurs : Ils fournissent un accès aux données Twitter relatives à un utilisateur particulier. Vous avez accès aux tweets, aux abonnés et aux amis d'un utilisateur de votre choix.
3. Flux de site : Les flux de sites sont utilisés par les serveurs qui se connectent à Twitter en tant que serveur proxy [4].

2. Python est un langage de programmation interprété
3. Tweepy est une librairie écrite en python, cette librairie permet à Python de communiquer avec la plateforme Twitter et d'utiliser son API.
4. De nombreux sites web utilisent ce point d'accès pour afficher les tweets dans une page web.

Paramètre	Description
Follow	Liste des utilisateurs auxquels renvoyer les statuts dans le flux.
Track	Mots-clés à suivre.
Locations	Spécifie une zone géographique à suivre.
Delimited	Spécifie si les messages doivent être séparés par une longueur.
Stall_warnings	Spécifie si les avertissements de décrochage doivent être livrés.

TABLE 2.1: Paramètres pour le point d'accès "POST statuses/filter"

Pour l'analyse en temps réel des données de Twitter, les flux publics constituent une bonne source d'API. Les flux publics sont divisés en trois points d'accès :

1. POST statuses/filter
2. GET statuses/sample
3. GET statuses/firehose

Le point de terminaison POST statuses/filter renvoie l'état associé aux divers paramètres de filtre. Plusieurs paramètres peuvent être fournis pour filtrer les requêtes, comme indiqué dans le tableau ci-dessus. Au moins un paramètre de filtre (Follow, Track, Locations) doit être mentionné afin de poursuivre le processus de captation. L'URL de la ressource permettant d'accéder à ce noeud final est la suivante :

https ://stream.twitter.com/1.1/statuses/filter.json

Différents paramètres peuvent être fournis comme suit : track = France & user = 123456789.

Le point de terminaison "GET statuses/sample" renvoie un petit échantillon de tous les statuts publics rassemblés au hasard. Ce point final est utile pour collecter des informations sur les sujets de tendances liés à un moment particulier. L'URL de la ressource est la suivante :

https ://stream.twitter.com/1.1/statuses/sample.json

Enfin, le point de terminaison Firehose renvoie tous les statuts publics. Cependant, une autorisation d'accès spéciale est requise par ce point d'accès final.

Ci-dessous nous présentons un exemple d'utilisation en Python de l'API streaming de Twitter en utilisant la librairie Tweepy.

```
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener

# consumer keys and access tokens, used for OAuth
consumer_key = 'YOUR-CONSUMER-KEY'
consumer_secret = 'YOUR-CONSUMER-SECRET'
access_token = 'YOUR-ACCESS-TOKEN'
access_token_secret = 'YOUR-ACCESS-TOKEN-SECRET'

class listener(StreamListener):

    def on_data(self, data):
        # print Json Twitter stream
        print(data)
        return(True)

    def on_error(self, status):
        print status

# OAuth process, using the keys and tokens
auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

# creation of the actual interface, using authentication
twitterStream = Stream(auth, listener())

# filter Twitter stream with some variables
# search tweets containing text "France"
# search french language tweets
twitterStream.filter(track=["France"], lang="fr")
```

Code 2.2: Exemple d'utilisation de l'API streaming en Python

2.2 Caractéristiques des données Twitter

Toutes les API Twitter qui renvoient des tweets fournissent les données codées à l'aide de la notation JSON [5]. Le format JSON est basé sur des paires clé-valeur, avec des attributs nommés et des valeurs associées. Ces attributs et leur état sont utilisés pour décrire les objets.

2.2.1 Structure

La figure ci-dessous est un exemple d'un tweet émis par l'API Twitter. Les informations fournies contiennent des données relatives au tweet : date de création du tweet, texte du tweet, adresses URL ou mentions d'utilisateur, etc. Des informations sur retweet sont également fournies avec les informations d'origines. Les informations concernant le créateur de la publication sont également associées aux informations de tweet.


```
"created_at": "Fri Jan 23 23:57:36 +0000 2015",
"id": 558775589612437504,
"id_str": "558775589612437504",
"text": "Thanks, polar vortex: Attendance dips at major Chicago museums in 2014 http:\/\/t.co\/jQlLEurk9s http:\/\/t.co\/3bss2nGemx",
"source": "\u003ca href=\"http:\/\/twitter.com\" rel=\"nofollow\"\u003eTwitter Web Client\u003c\/a\u003e",
"truncated": false,
"in_reply_to_status_id": null,
"in_reply_to_status_id_str": null,
"in_reply_to_user_id": null,
"in_reply_to_user_id_str": null,
"in_reply_to_screen_name": null,
"user": {
  "id": 7313362,
  "id_str": "7313362",
  "name": "Chicago Tribune",
  "screen_name": "chicagotribune",
  "location": "Chicago, IL",
  "url": "http:\/\/www.chicagotribune.com\/",
  "description": "Chicago Tribune news, features and so much more live from our newsroom. A part of your life since 1847.",
  "protected": false,
  "verified": true,
  "followers_count": 321548,
  "friends_count": 523,
  "listed_count": 8074,
  "favourites_count": 34,
  "statuses_count": 47367,
  "created_at": "Sat Jul 07 14:10:07 +0000 2007",
  "utc_offset": -21600,
  "time_zone": "Central Time (US & Canada)",
  "geo_enabled": false,
  "lang": "en",
```


FIGURE 2.3: Exemple de Twitter JSON

2.2.2 Contenus

Sur Twitter, nous recevons de nombreux objets, y compris les tweets et les utilisateurs. Ces objets encapsulent tous les attributs principaux qui décrivent l'objet. Chaque tweet comporte un auteur, un message, un identifiant unique, un horodatage de publication et parfois des métadonnées géographiques partagées par l'utilisateur. Chaque utilisateur a un nom Twitter, un identifiant, un nombre d'abonnés et le plus souvent une biographie de

5. JavaScript Object Notation (JSON) est un format de données textuelles dérivé de la notation des objets du langage JavaScript.

compte.

Avec chaque tweet, nous générons également des objets 'entité', qui sont des tableaux de contenus de Tweet communs tels que des hashtags, des mentions, des médias et des liens. S'il existe des liens, la charge JSON peut également fournir des métadonnées telles que l'URL entièrement déroulée, ainsi que le titre et la description de la page Web. Ainsi, en plus du contenu du texte, un tweet peut avoir plus de 150 attributs associés. Reflétant la hiérarchie JSON ci-dessus, voici une descriptions supplémentaires de ces objets :

1. Tweet : Également appelé objet 'Statut', possède de nombreux attributs de niveau racine, parent de d'autres objets.
   (a) Utilisateur : Métadonnées au niveau du compte Twitter. Inclut tous les enrichissements disponibles au niveau du compte, tels que Profile geo.
   (b) Entités : Contient jusqu'à quatre photos natives, ou une vidéo ou un fichier GIF animé.
   (c) Entités étendues : Contient les tableaux d'objets de #hashtags, @mentions, $ symbol, URL et media.
   (d) Lieux : Parent à l'objet 'coordonnées' .

Lors de l'acquisition de données Tweet, l'objet principal est l'objet Tweet, qui est un objet parent de plusieurs objets enfants. Par exemple, tous les Tweets incluent un objet Utilisateur qui décrit l'auteur du Tweet. Si le tweet est géolocalisé, un objet « lieu » est inclus. Chaque tweet comprend un objet « entités » qui encapsule des tableaux de hashtags, de mentions utilisateur, d'URL et de médias natifs. Si le Tweet contient un média « attaché » ou « natif » (photos, vidéo, GIF animé), il y aura un objet « extended_entities ».

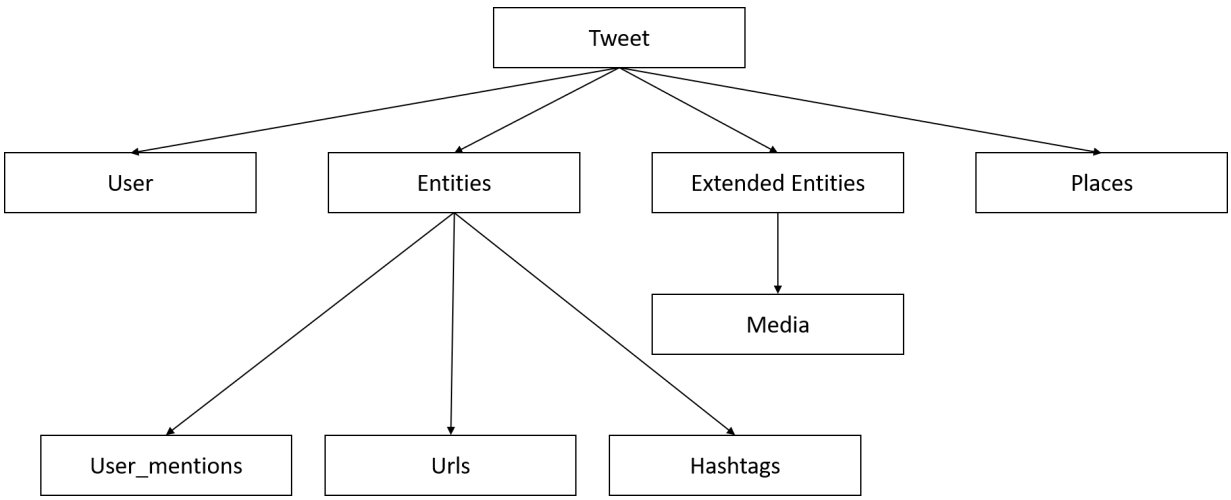

FIGURE 2.4: Exemple d'arborescence Twitter

2.3 Construction d'un graphe social à partir d'un arbre JSON

Nous distinguons deux grands types de graphes possibles sur Twitter, tout d'abord nous avons le graphe classique d'utilisateur. Ce graphe modélise sous forme de réseau les relations entre les utilisateurs 2.1. Une autre approche consiste à créer un graphe à partir d'événements ; nous définissions ici un événement comme étant l'activité générée par l'émission de tweets. Lors de l'émission d'un tweet, un certain nombre d'informations sont produites. Ces informations peuvent être également représentées sous forme de graphe. Pour ce dernier type de graphe, de nombreux modèles existent. Nous avons étudié deux d'entre eux.

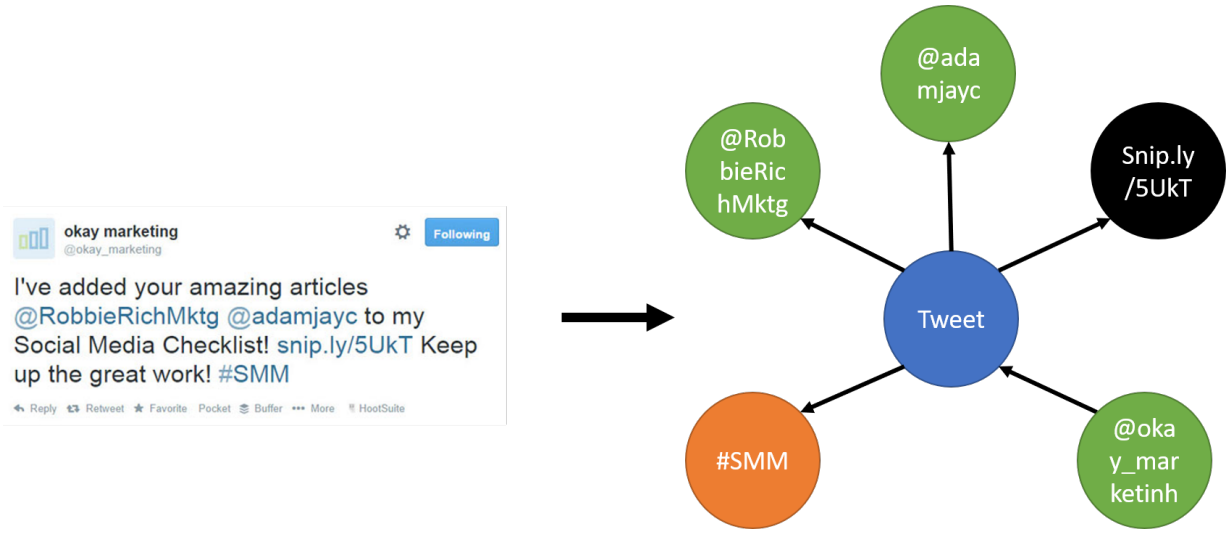

FIGURE 2.5: D'un flux JSON à un graphe

2.3.1 Modèle Naoyun Gephi

Dans le modèle Naoyun [6] (figure 2.6), un graphe dirigé G est généré à partir des tweets. Ce modèle dispose de 4 types de nœuds :

1. *Tweet* Le tweet ;
2. *@User* L'auteur du tweet ou les mentions d'utilisateurs dans le tweet ;
3. *#hashtags* Les mots clés présents dans le tweet ;
4. *Http* Les urls présentes dans le tweet.

6. Naoyun est un logiciel qui permet de créer un pont entre l'API Twitter et l'outil de visualisation de graphe Gephi.

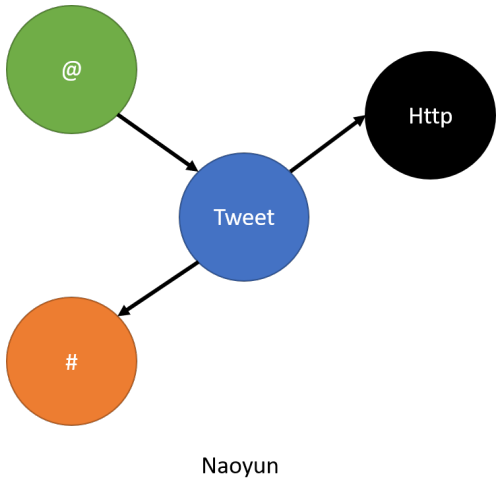

FIGURE 2.6: Exemple d'un modèle de graphe d'évènements Naoyun sur Twitter

Dans ce modèle, il existe une arête entre :

— L'auteur du tweet $@User$ et le $Tweet$;
— Le $Tweet$ et les mentions $@User$ présents dans le tweet ;
— Le $Tweet$ et les mots clés $\#hashtags$ présents dans le tweet ;
— Le $Tweet$ et les urls $Http$ présents dans le tweet ;

En cas de retweet RT l'auteur du retweet est connecté avec le $Tweet$ d'origine et son auteur $@User$.

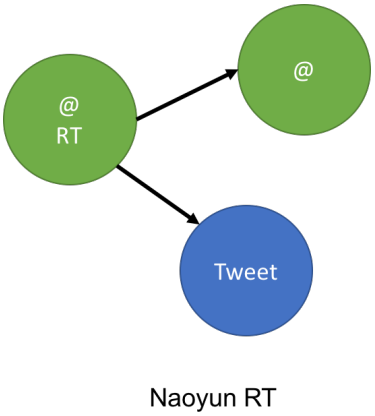

FIGURE 2.7: Exemple d'un modèle de graphe d'événements Naoyun sur Twitter avec un retweet RT

Nous allons générer un graphe G avec le modèle naoyun à partir d'une séquence de tweets présente dans le tableau 2.2.

Tweet_id	Auteur	Contenu
1	@Gagarine	I see #earth from #vostok ! http ://foto.com/AzeRT
2	@USSR	Yes we've done it ! #vostok http ://foto.com/AzeRT
3	@USSA	OMG ! @Gagarine wen't to space ! #fear
4	@Baikonour	RT @Gagarine I see #earth from #vostok ! http ://foto.com/AzeRT

TABLE 2.2: Exemple de tweets

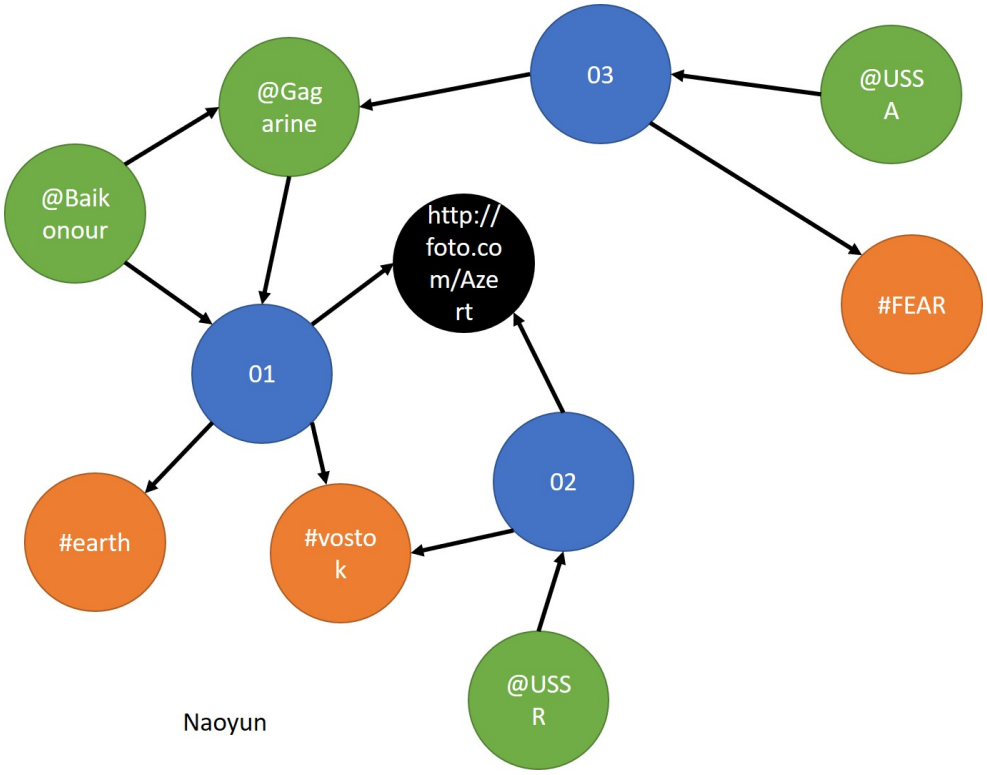

FIGURE 2.8: Exemple d'un graphe G naoyun généré à partir de tweets

2.3.2 Twitter graphe d'un flux de données

Dans ce modèle (figure 2.9), un graphe G orienté est généré à partir du contenu des tweets. Ce modèle dispose par défaut de 2 types nœuds, $@user$ et $\#hashtags$ mais nous pouvons l'étendre aux urls *http* et aux *images*. Dans notre modèle, il existe une arête entre :

- L'auteur $@User$ et les mentions $@User$ présentes dans le tweet ;
- L'auteur $@User$ et les mots clés $\#hashtags$ présents dans le tweet.
- En cas de retweet RT :
  - L'auteur du retweet $@User$ est connecté à l'auteur du tweet d'origine
  - L'auteur du retweet $@User$ et les mots clés $\#hashtags$ présents dans le tweet.

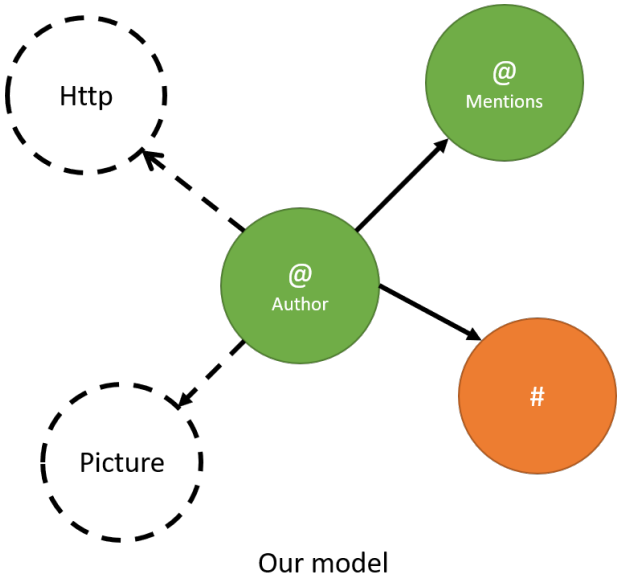

FIGURE 2.9: Notre modèle de graphe

Nous allons générer un graphe G avec notre modèle, à partir de la séquence de tweet présente dans le tableau 2.2.

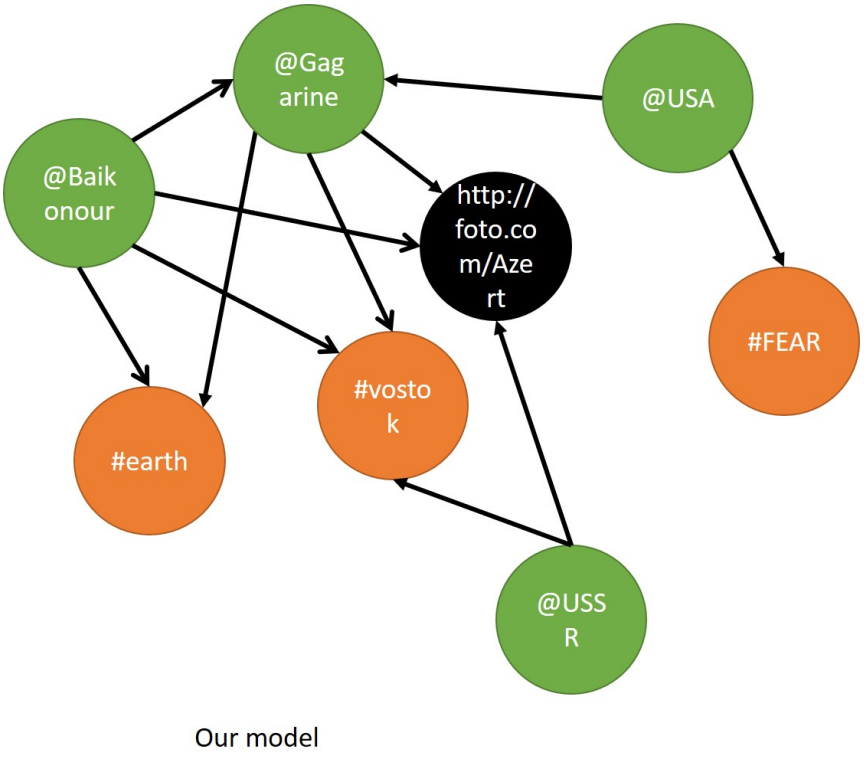

FIGURE 2.10: Exemple d'un graphe G avec notre modèle, généré à partir de tweets

La différence majeure entre ces deux différents modèles de graphes d'événements réside sur la prise en compte du *Tweet* comme un nœud dans le graphe. Le modèle "Twitter graphe" met en avant le contenu et non le contenant. Lors de la création du graphe, cette distinction joue un rôle important si nous devons approximer les arêtes d'un graphe avec des méthodes basées à partir d'un réservoir, que nous verrons dans les chapitres suivants.

```
#data is a twitter JSON tree
def create_graph(data):
    all_data = dict(data)
    edge_list = []
    user_screen_name = all_data['user']['screen_name']
    if len(all_data['entities']['hashtags']) > 0:
        for index, i in enumerate(all_data['entities']['hashtags']):
            edge_list.append((user_screen_name, all_data['entities']['hashtags'][index]['text']))
    if len(all_data['entities']['user_mentions']) > 0:
        for index, i in enumerate(all_data['entities']['user_mentions']):
            edge_list.append((user_screen_name, all_data['entities']['user_mentions'][index]['screen_name']))
    return edge_list
```

Code 2.3: Exemple de création d'un graphe à partir d'un arbre JSON avec notre modèle

2.4 Propriétés d'un graphe Twitter

Un réseau social est défini comme une représentation des interactions sociales au sein d'un groupe d'individus. Le réseau social est le plus souvent visualisé sous forme de graphe avec des individus comme des nœuds et des relations comme des arêtes. Les arêtes peuvent être dirigées, représentant par exemple une mention "@" d'une personne à une autre ou non-dirigées, représentant par exemple une relation. Les contacts et les relations ne doivent pas nécessairement être représentés sous forme d'arêtes, nous pouvons créer un graphe d'événements (actions et relations) où est représenté, sous forme de nœuds et arêtes, un graphe qui modélise les interactions produites dans le réseau.

L'analyse des réseaux sociaux utilise plusieurs concepts issus de la théorie des graphes dans ses algorithmes et métriques. La théorie des graphes nous fournit des outils pour représenter formellement les réseaux sociaux et quantifier les propriétés structurelles du réseau. Nous présentons ci-dessous les propriétés d'un graphe type issu du réseau social Twitter.

2.4.1 Distribution des degrés

La figure ci-dessous, illustre la distribution en loi de puissance des degrés dans un graphe type issu de Twitter. La probabilité pour un nœud donné d'avoir k arêtes est définie comme suit $P(k) \sim k^{-\lambda}$.

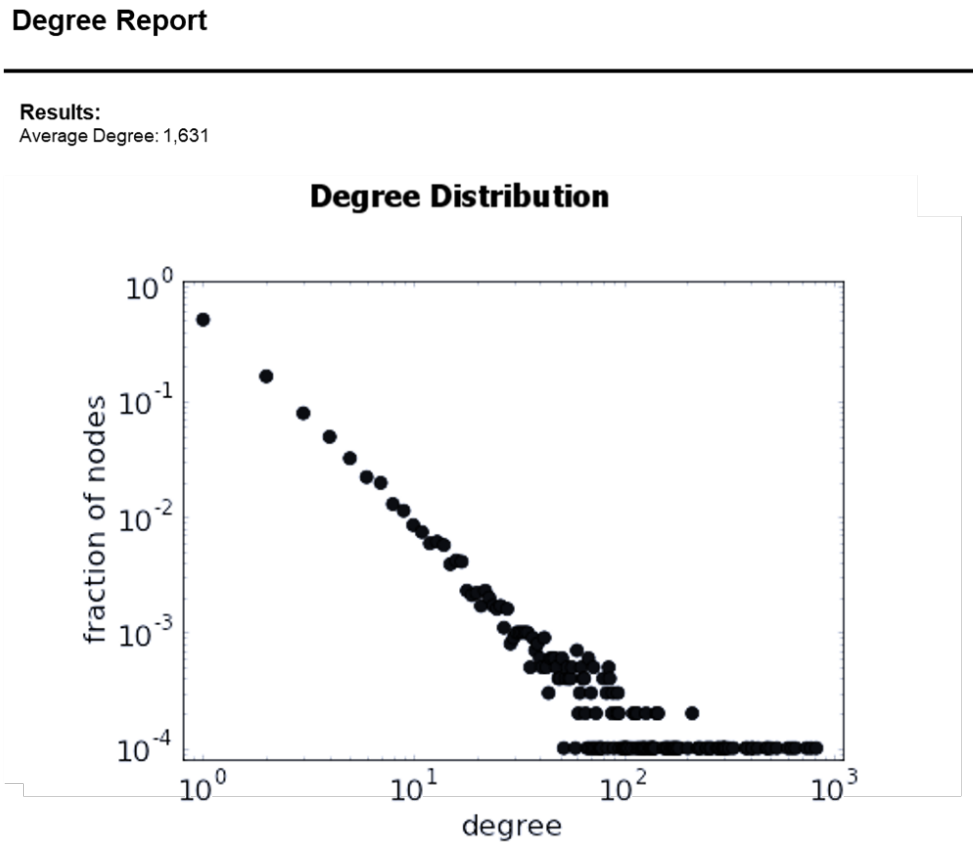

FIGURE 2.11: Exemple de la distribution des degrés dans un graphe type Twitter, sur un repère log-log

Les réseaux possédant cette propriété sont appelés réseaux sans échelle [9]. L'équation1.7 implique que les réseaux sans échelle ont très peu de nœuds hautement connectés, alors que la majorité d'entre eux ne le sont pas. Les nœuds fortement connectés sont appelés hubs et leurs degrés ont tendance à être plus élevés que ceux des nœuds moyens.

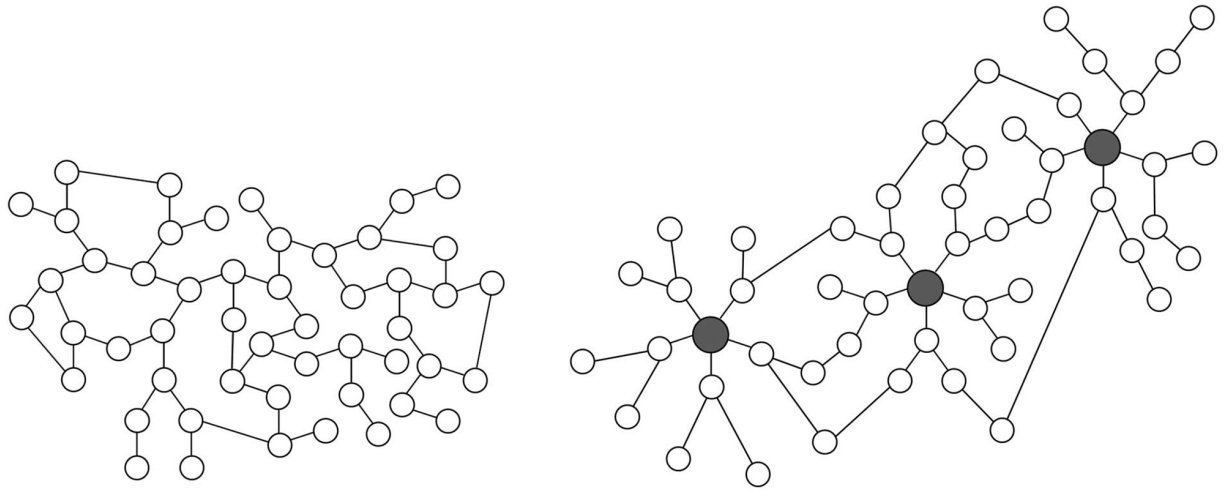

FIGURE 2.12: Exemple de réseau sans échelle

Twitter peut être classé comme un réseau sans échelle. La manière dont les réseaux sans échelle se développent peut s'expliquer par la croissance basée sur l'attachement préférentiel [10]. Selon cette hypothèse, à mesure que le réseau se développe, les nouveaux nœuds sont plus susceptibles de se connecter à des nœuds de haut niveau qu'à des nœuds de faible degré. Une analogie avec l'effet Matthieu peut être faite, dans l'idée que « les riches s'enrichissent et les pauvres s'appauvrissent ». Cet effet est observable sur le réseau Twitter, puisque les nouveaux utilisateurs ont tendance à suivre les utilisateurs importants (c'est-à-dire hautement connectés).

2.4.2 Diamètre et centralité

Le concept de diamètre d'un graphe est important car il quantifie la distance entre les deux sommets les plus éloignés d'un graphe. Le diamètre D d'un graphe $G = (V, E)$ est égal à la valeur maximale de l'excentricité parmi toutes paires de sommets du graphe. En d'autres termes, c'est la plus grande distance entre deux paires de sommets dont le diamètre peut varier dans l'intervalle $[1, V - 1]$. Même si dans un graphe déconnecté le diamètre est infini, il est possible de trouver le diamètre de la plus grande composante connectée, ainsi que celui d'un sous-graphe.

Nous appliquons à un graphe G issu de Twitter l'algorithme de Brandes [16], nous obtenons les résultats ci-dessous.

Diamètre	3
Radius[1]	1
Distance moyenne des chemins	1.8331587218438974

"Betweenness Centrality" mesure la fréquence d'apparition d'un nœud sur les chemins les plus courts entre les nœuds du réseau.

"Closeness Centrality" correspond à la distance moyenne depuis un nœud de départ vers tous les nœuds du réseau.

"Eccentricity" correspond à la distance depuis un nœud de départ vers le nœud le plus éloigné dans le réseau.

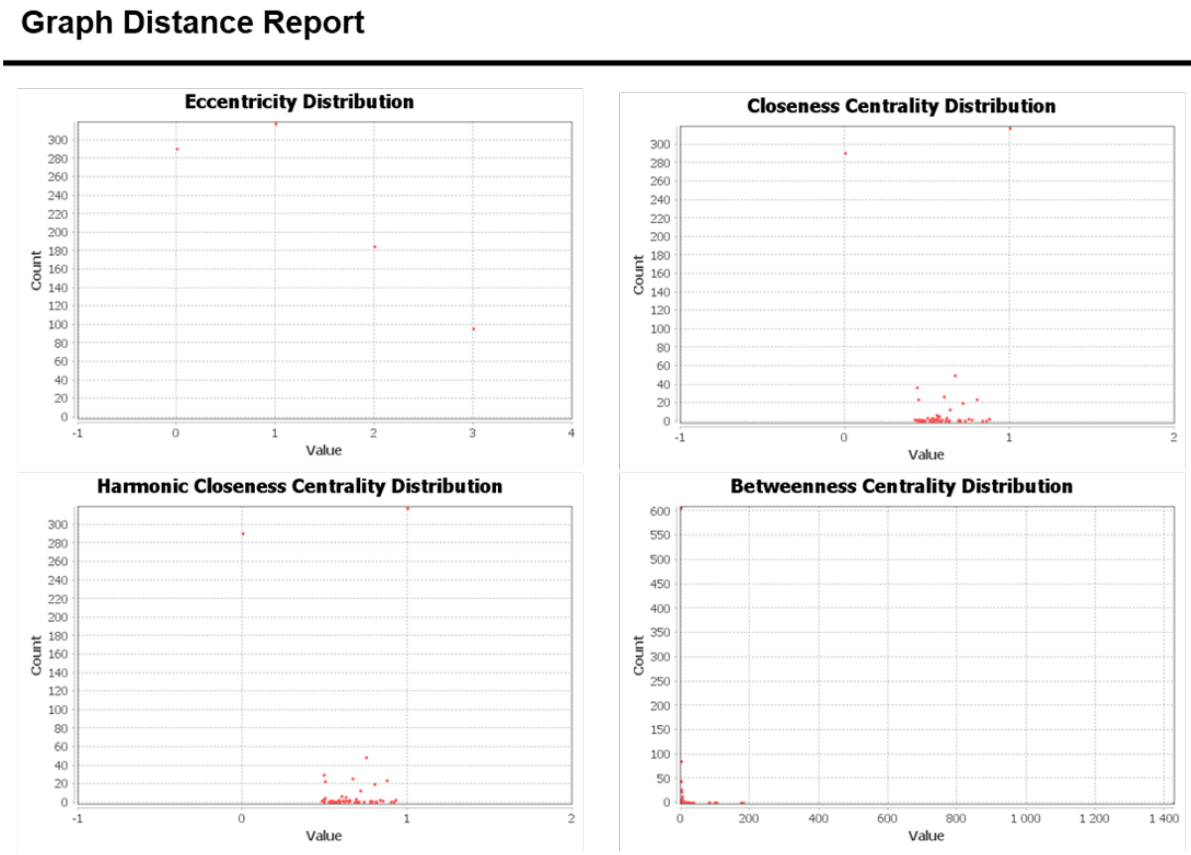

FIGURE 2.13: Diamètre d'un graphe G Twitter

1. Le rayon d'un graphe est l'excentricité minimale de ses sommets, c'est-à-dire la plus petite distance à laquelle puisse se trouver un sommet de tous les autres

2.4.3 Distance moyenne des chemins

Les réseaux du monde réel entrent généralement dans la catégorie des réseaux des "petits mondes". Les réseaux des petits mondes se caractérisent par le fait que leur longueur de chemin moyenne soit très faible par rapport à la taille du réseau. Cette propriété intéressante n'est cependant pas si rare, car elle se trouve également dans des graphes aléatoires [61].

Au cours des dernières années, de nombreuses études ont été menées pour analyser la structure topologique complexe de plusieurs réseaux réels. Le World Wide Web en est un exemple : les pages Web forment l'ensemble des nœuds du réseau et les liens hypertextes entre eux sont les arêtes. Malgré le grand nombre de pages Web sur Internet et les liens entre elles, sa structure montre une longueur de chemin moyenne très faible, un coefficient de clustering élevé et une distribution des degrés basée sur la loi de puissance.

3 Méthodes de détection de clusters dans un graphe

Les réseaux sociaux présentent une structure dans laquelle le réseau est constitué de sous-ensembles de nœuds hautement connectés. Ils ont relativement peu de connexions, des noeuds extérieurs aux autres sous-sembles. Les nœuds d'un tel sous-ensemble sont susceptibles de partager des attributs ou des propriétés communes, formant ainsi en extrapolant, une communauté. Il est donc très intéressant de pouvoir trouver des méthodes afin de détecter ces structures.

Trouver ces structures de manière efficace n'est pas toujours trivial. Nous retenons deux approches principales en matière de détection de cluster, le partitionnement et la classification hiérarchique.

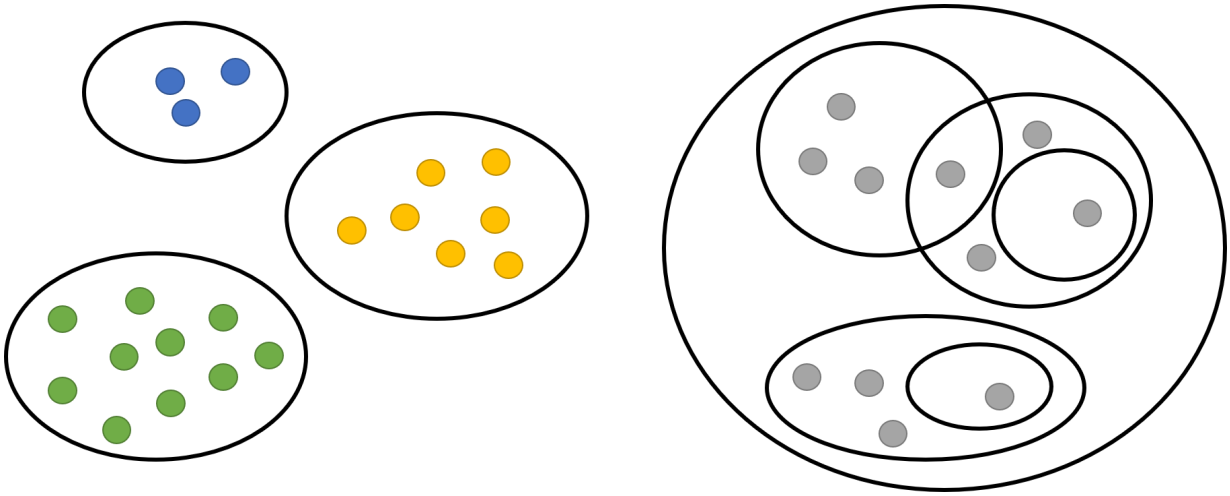

FIGURE 3.1: À gauche illustration d'un partitionnement, chaque nœud est assigné à un cluster unique. À droite illustration d'une classification hiérarchique de partitions

Partitionnement. Le partitionnement, également appelé clustering à plat, crée un ensemble plat de clusters sans structure explicite qui pourrait relier les clusters les uns aux autres. Les méthodes de regroupement à plat sont conceptuellement simples, mais elles présentent un certain nombre d'inconvénients. La plupart des algorithmes de partitionnement, nécessitent un nombre prédéfini de clusters en entrée et ne sont pas déterministes. Le regroupement à plat peut être considéré comme un problème d'optimisation.

Regroupement hiérarchique. Le regroupement hiérarchique crée une hiérarchie de clusters ou en d'autres termes, une arborescence de clusters. La sortie hiérarchique est donc plus structurée que le clustering à plat.

Dans ce chapitre, nous étudions différentes techniques de détections de clusters, également appelées détection de communautés. Sur la base de l'approche utilisée, les techniques de clustering peuvent être classées dans les méthodes de calcul suivantes : agglomération, division et spectral.

— *Les techniques d'agglomérations* il s'agit d'une approche « ascendante » : chaque observation commence dans son propre cluster et des paires de clusters sont fusionnées au fur et à mesure que l'on monte dans la hiérarchie.

— *Les techniques de divisions* il s'agit d'une approche « descendante » : toutes les observations commencent dans un cluster et les scissions sont effectuées de manière récursive au fur et à mesure que l'on descend dans la hiérarchie.

— *Les techniques spectrales* il s'agit de diviser le graphe en structures de communautés basées sur les valeurs propres / vecteurs propres du Laplacien du graphe.

Une étape cruciale dans tout algorithme d'identification des structures de communauté consiste à sélectionner des métriques appropriées pour mesurer la similarité ou la dissimilarité entre les nœuds. L'objectif reste de regrouper des données similaires, ce qui constituerait une communauté en utilisant des mesures de similarités.

3.1 Techniques d'agglomérations

3.1.1 Clustering agglomératif

Le clustering agglomératif recherche les clusters en définissant initialement chaque nœud comme son propre cluster. L'étape suivante consiste à fusionner la paire de nœuds la plus similaire, en utilisant certains critères tels que la mise en cluster à liaison unique, à liaison complète ou à liaison moyenne :

— *Cluster à liaison unique.* Le clustering à liaison unique, également appelé méthode du plus proche voisin, peut être utilisé à la fois avec des mesures de similarité et avec des mesures de distance. Les groupes sont fusionnés en fonction de la distance entre

leurs membres les plus proches. Le clustering à liaison unique fusionne les paires de clusters ayant le chemin le plus court possible.

— *Cluster à liaison complète.* Le clustering à liaison complète, également appelé méthode du voisin le plus éloigné, est l'opposé de la méthode du clustering à liaison unique. En effet, la distance entre les groupes est définie comme la distance entre leurs paires les plus éloignées. La liaison complète fusionne donc les deux groupes pour lesquels les nœuds des groupes ont le plus petit diamètre lorsqu'ils sont fusionnés.

— *Cluster à liaison moyenne.* Le clustering des liens moyens, également appelé UPGMA (Unweighted Pair-Group Method using Arithmetic averages), fusionne les clusters qui ont la distance moyenne la plus courte pour toutes les paires de nœuds, où les nœuds de la paire ne proviennent pas du même cluster. Cette méthode constitue un compromis entre les méthodes de couplage simple et complète.

Ces regroupements sont répétés jusqu'à ce que tous les nœuds soient regroupés en un seul cluster ou lorsque l'on a atteint le nombre de clusters ciblés.

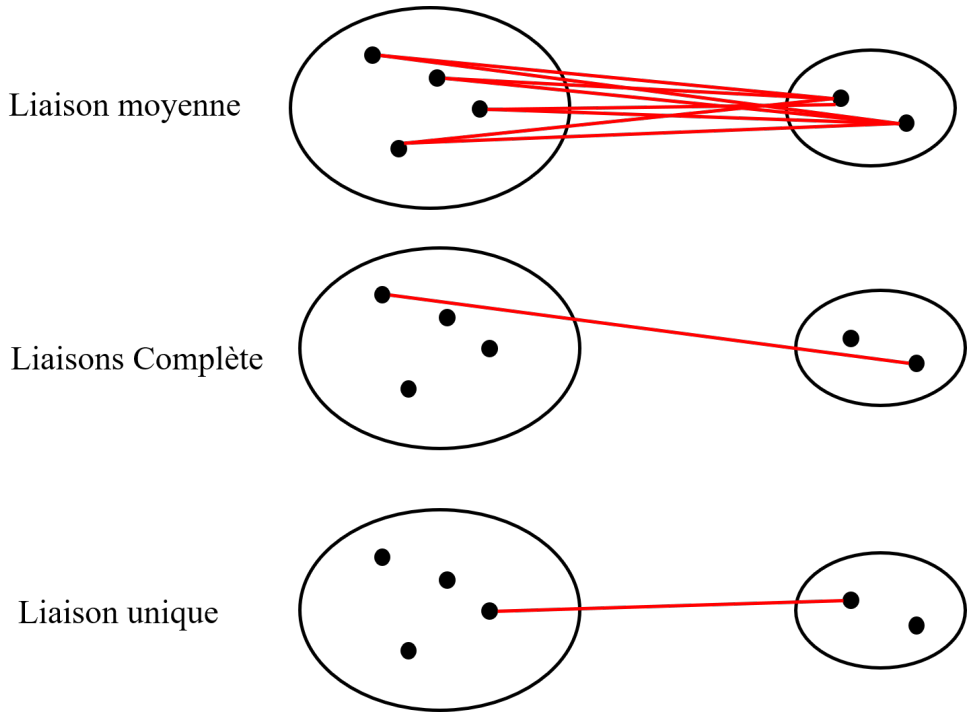

FIGURE 3.2: Méthodes de clustering agglomératif

3.1.2 k-means sur les graphes "spatialisés"

L'algorithme de regroupement k-means est connu pour être efficace dans le regroupement de grands ensembles de données. Cet algorithme est l'un des algorithmes d'apprentissage non-supervisé le plus simple et le mieux connu. L'algorithme k-means vise à partitionner un ensemble d'objets, en fonction de leurs attributs / caractéristiques, en k clusters, où k est une constante prédéfinie.

Si nous ne disposons pas d'un graphe d'attribut, nous pouvons représenter notre graphe G en 2 dimensions avec des coordonnées (x, y) et pour cela, certains algorithmes de spatialisation dirigés peuvent être utilisés.

Par exemple, le ForceAtlas2 [40] est un algorithme de spatialisation dirigé par la force : il simule un système physique afin de spatialiser un graphe. Les nœuds se repoussent comme des aimants, tandis que les arêtes attirent leurs nœuds comme des ressorts. Ces forces créent un mouvement qui converge vers un état équilibré.

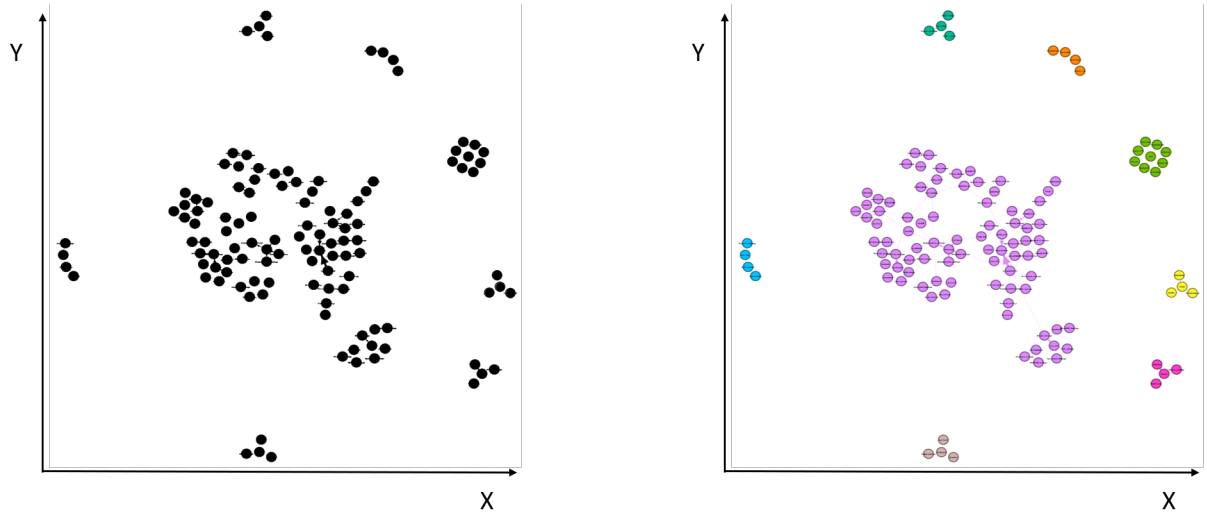

FIGURE 3.3: Représentation d'un graphe en 2 dimensions

L'algorithme définit k centroïde, un pour chaque cluster. Le centroïde d'un cluster est formé de manière à ce qu'il soit étroitement lié, en termes de similarité (où la similarité peut être mesurée en utilisant différentes méthodes telles que la distance euclidienne ou de jacquard) à tous les objets de ce groupe. Techniquement, ce qui est intéressant dans le k-means, c'est la variance.

Cela permet de minimiser la variance globale en affectant chaque objet au cluster. Les étapes de base de l'algorithme de regroupement k-means sont les suivantes :

1. Choisir k objets au hasard et laissez-les définir k clusters.
2. Calculez les représentants du cluster.
3. Créez de nouveaux groupes, un par représentant de cluster. Laissez chaque texte appartenir au cluster avec le représentant de cluster le plus similaire
4. Répétez l'étape *2* jusqu'à ce qu'un critère d'arrêt soit atteint.

La première étape définit une partition initiale aléatoire. Il existe de nombreuses autres manières de le construire et le résultat dépend de celui qui est utilisé. En tant que représentant de cluster, la moyenne (le centroïde) des objets du cluster est généralement utilisée. D'autres variantes consistent à laisser la médiane ou quelques objets spécifiques représenter le cluster.

Lorsque le regroupement est flou, les objets peuvent appartenir à plusieurs clusters et par conséquent, le représentant du cluster peut être calculé en tenant compte de cela. Le critère d'arrêt normal est lorsque aucun objet ne change de cluster ou lorsque très peu de clusters changent entre les itérations. Il peut également s'agir de s'arrêter après un nombre prédéfini d'itérations, car la plupart des améliorations de la qualité sont généralement obtenues lors des premières itérations.

La complexité temporelle de l'algorithme k-means est $O(knI)$, où k est le nombre de clusters, n le nombre d'objets et I le nombre d'itérations (qui dépend du critère d'arrêt). Dans chaque itération, les représentants du cluster et les similarités k_n entre tous les objets et tous les groupes doivent être calculés. L'algorithme k-means nécessite un certain nombre de clusters en entrée. C'est-à-dire qu'il faut deviner le nombre approprié de clusters.

Dans un algorithme de partitionnement général, le fractionnement et la division des clusters sont autorisés et, théoriquement, le résultat est le nombre optimal de clusters.

3.2 Techniques de division

3.2.1 Girvan et Newman

S'il existe des communautés dans un réseau social, on peut supposer qu'elles sont connectées par un nombre relativement restreint d'arêtes. En supprimant ces arêtes, nous pouvons trouver la structure de la communauté dans le réseau.

Girvan et Newman ont utilisé une mesure introduite dans [36] connue sous le nom de « betweenness centrality » notée B pour chaque arête. La « betweenness centrality » d'une arête est définie comme le nombre de plus courts chemins entre les paires de nœuds du graphe qui passent par l'arête. Si une paire de nœuds a plusieurs plus courts chemins

entre eux, alors chaque chemin est pondéré par la somme de tous ses chemins. Ainsi, la « betweenness centrality » pour une arête e peut être définie [37] comme suit :

$$B(e) = \sum_{i \in V, j \in V} \frac{\sigma_e(i,j)}{\sigma(i,j)} \tag{3.1}$$

Où $\sigma(i,j)$ représente le nombre de plus courts chemins entre i et j, et $\sigma_e(i,j)$ représente le nombre de plus courts chemins entre i et j qui inclut e.

Contrairement à la classification hiérarchique, l'algorithme de Girvan Newman recalcule les limites de chaque étape. L'algorithme fonctionne comme suit :

1. Calculer la centralité (betweenness) $B(e)$ pour toutes les arêtes
2. Retirer l'arête avec la plus grande distance
3. Recalculer la distance de toutes les arêtes restantes
4. Répéter jusqu'à ce qu'il ne reste plus d'arêtes

On peut voir cet algorithme comme un algorithme de classification hiérarchique descendant (divisant). La racine du dendrogramme regroupe tous les nœuds en une seule communauté. Chaque branche de l'arborescence représente l'ordre de répartition du réseau à mesure que les arêtes sont supprimées.

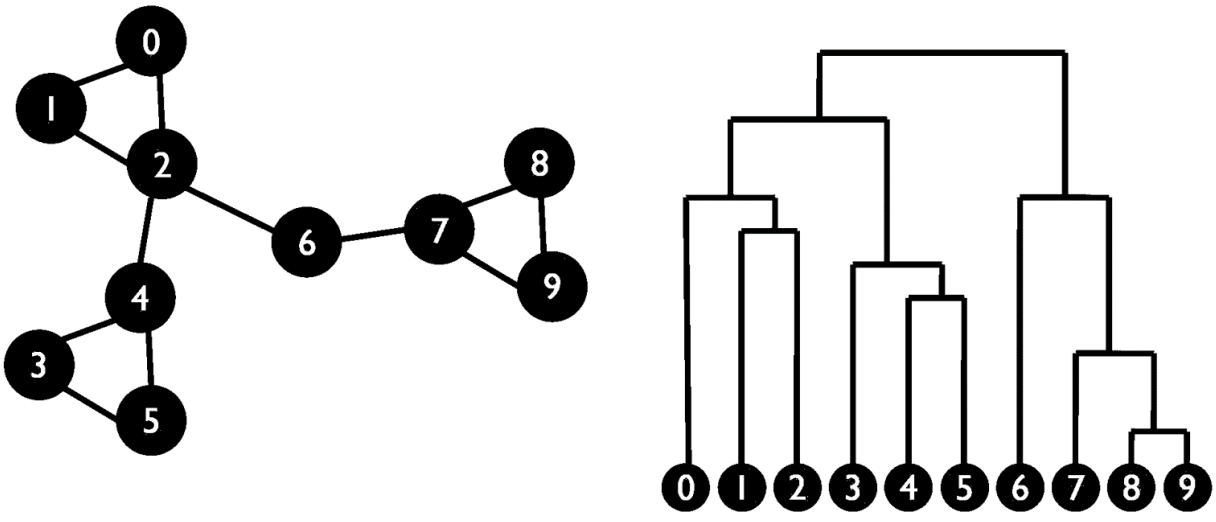

FIGURE 3.4: Méthodes de clustering agglomératives

3.2.2 Heuristique de Kernighan-Lin

Étant donné une division initiale d'un graphe en deux sous-graphes, l'algorithme Kernighan-Lin [42] essaie de minimiser le nombre d'arêtes entre les sous-graphes en échangeant des paires de nœuds. Kernighan-Lin calcule la valeur la plus élevée possible pour la fonction de bénéfice, en échangeant un noeud d'un sous-graphe avec un nœud de l'autre sous-graphe. La fonction de bénéfice est définie comme le nombre d'arêtes à l'intérieur des sous-graphes, soustraites par le nombre d'arêtes entre les sous-graphes.

La paire de nœuds qui donne la valeur la plus élevée à la fonction de bénéfice est alors échangée, avec la restriction que chaque paire ne peut être échangée qu'une seule fois. Une fois que tous les nœuds d'un sous-graphe ont été échangés, nous notons après quel swap la fonction de bénéfice a atteint sa valeur maximale et divise le graphe en conséquence.

Cependant, le problème avec Kernighan-Lin est que nous devons connaître la taille des sous-graphes à l'avance, car les mauvaises tailles donnent des résultats non-optimaux [51], ce qui rend l'algorithme inadapté à la plupart des réseaux sociaux.

3.3 Techniques spectrales

3.3.1 Partitionnement spectral

Les algorithmes de classification spectrale utilisent le plus souvent l'information contenue dans les vecteurs propres d'une matrice Laplacienne notée L. Celle-ci est construite à partir de la matrice d'adjacence A. Les algorithmes de classification spectrale trouvent des clusters en factorisant la matrice avec les valeurs propres et en regroupant les vecteurs propres.

Pour définir une matrice d'adjacence A, il faut une méthode pour mesurer la similarité a_{ij} pour les nœuds u_i et u_j. Deux nœuds, u_i et u_j, sont connectés si la valeur de a_{ij} est supérieure à une valeur de seuil. Nous voulons trouver des clusters dans le graphe de similarité pour lesquels, les arêtes entre les clusters ont des valeurs faibles et pour lesquels, les arêtes au sein d'un cluster ont des valeurs élevées.

Les graphes de similarités courants sont :

— *Graphe de voisinage* ϵ pour lesquels nous connectons les nœuds avec des distances supérieures à ϵ.
— *Graphe des k plus proches voisins* pour lesquels nous connectons un nœud u_i avec un nœud u_j si u_j est l'un des k-nœuds les plus proches de u_i.
— *Graphe totalement connecté* pour lesquels nous connectons tous les nœuds avec une similitude positive les uns avec les autres, et nous pondérons toutes les arêtes par a_{ij}.

La matrice Laplacienne L d'un graphe est définie comme :

$$L = D - A \tag{3.2}$$

où D est la matrice diagonale des degrés telle que :

$$D_{ii} = \sum_{j=1}^{N} A_{ij} \tag{3.3}$$

représente le degré du i^e nœud du graphe G.

Connaissant les vecteurs propres, nous pouvons projeter ces vecteurs propres dans une dimension k où k est le nombre de valeurs propres que nous avons gardées. Ainsi, le partitionnement peut se faire de différentes manières ; nous pouvons utiliser par exemple le k-means sur les vecteurs associés à chacun des nœuds pour affecter chaque nœud à une classe. Ces classes représentent les communautés auxquelles les individus ont été assignés.

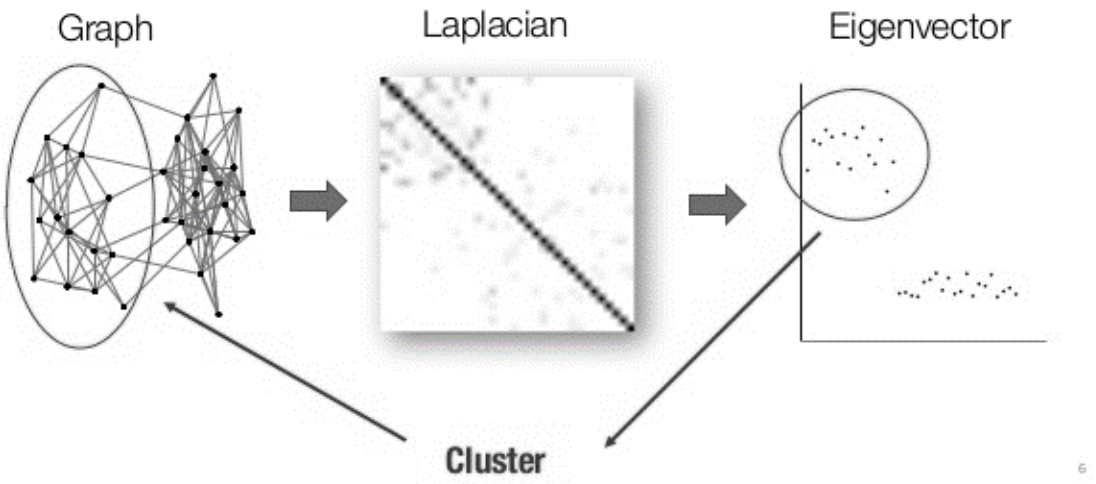

FIGURE 3.5: Illustration du partitionnement spectral, source : Austin Benson, Higher-order graph clustering at AMS Spring Western Sectional

3.3.2 Modularité

> "refers to a group of physically or functionally linked nodes that work together to achieve a distinct funcfion" (Barabasi & Oltvai)

La modularité permet de mesurer la qualité d'un partitionnement d'un graphe. Si nous avons trouvé des clusters ayant beaucoup moins de connexion entre les clusters, ou ayant beaucoup plus de connexion entre les nœuds d'un même graphe que celles d'un graphe aléatoire, nous avons probablement trouvé une bonne division. Newman et Girvan ont introduit le concept de modularité comme mesure de la qualité d'une partition [52].

Si nous divisons un graphe en k clusters, nous pouvons créer une $k * k$ matrice e dans laquelle e_{ij} est la fraction des arêtes reliant un cluster i et j. La trace $tr(e)$ est $\sum_{i=1}^{k} = e_{ii}$ qui donne la fraction d'arêtes entre les nœuds d'un même cluster i. Une valeur élevée de $tr(e)$ indique une bonne division. a_i donne le nombre d'arêtes dans le cluster i.

En utilisant cela, nous pouvons définir une mesure de modularité Q comme :

$$Q = \sum_{i=1}^{k} = (e_{ii} - a_i^2) \tag{3.4}$$

a_i est élevé au carré pour représenter la probabilité qu'une arête aléatoire tombe dans le module i. Une valeur de Q proche de 0 indique une mauvaise division et une valeur de Q proche de 1 indique une bonne division [52].

Nous pouvons maximiser Q en utilisant une approche spectrale ; A_{ij} sont définis comme les éléments de la matrice adjacence A, et s est le vecteur de la colonne représentant toute division du réseau en deux groupes. Ces éléments sont définis comme $s_i = +1$ si le nœud i appartient au premier groupe et $s_i = -1$ s'il appartient au second groupe. B est la matrice de modularité qui comporte des éléments.

$$B_{ij} = A_{ij} - \frac{k_i k_j}{2m} \tag{3.5}$$

Toutes les lignes et colonnes de la matrice de modularité B totalisent zéro, ce qui signifie que la modularité d'un réseau non divisé est toujours nulle.

$$Q = \frac{1}{4m} \sum_{ij} (A_{ij} - \frac{k_i k_j}{2m}) s_i s_j = \frac{1}{4m} s^T B s \tag{3.6}$$

4 Méthodes d'approximation de clusters dans un graphe social statique

Dans ce chapitre nous introduisons des méthodes d'approximation pour la détection et l'estimation de clusters. Un principe général est d'échantillonner les arêtes d'un graphe G uniformément, comme dans le modèle d'Erdös-Renyi. Contrairement au modèle d'Erdös-Renyi nous proposons un modèle qui ne part pas du graphe complet mais d'un graphe social qui suit une distribution de degrés selon une loi de puissance. Nous combinons donc le modèle de configuration pour générer G, puis appliquons le modèle Erdös-Renyi pour obtenir un graphe aléatoire en sélectionnant certaines arêtes.

L'algorithme de détection de cluster est probabiliste, car la sélection de chaque arête est probabiliste : on définit un espace probabiliste Ω des sous-graphes possibles, chacun obtenu avec la même probabilité. Les notions d'approximation sont les suivantes :

— Pour un problème de décision soit L l'ensemble des instances positives. Un algorithme d'approximation A doit satisfaire les deux conditions :

$$Si\ x \in L,\ alors\ Prob_{\Omega}[A(x)\ accepte]\ \geq 1-\delta$$

$$Si\ x \notin L,\ alors\ Prob_{\Omega}[A(x)\ rejette]\ \geq 1-\delta$$

— Pour un problème de calcul, estimer un ensemble C par exemple, un algorithme d'approximation A produit un ensemble C' que l'on peut comparer à C à l'aide d'une distance $dist(C, C')$. On note $A(x) = C'$, et A doit satisfaire la condition :

$$Prob_{\Omega}[dist(A(x), C) \leq \epsilon]\ \geq 1-\delta \tag{4.1}$$

Dans ce cas, on utilise deux paramètres ϵ et δ.

L'étude se concentre sur l'existence de composantes géantes dans ce nouveau modèle (modèle de configuration qui suit une loi de puissance suivi d'Erdös-Renyi), et l'analyse de ces composantes nous permet d'estimer les clusters du graphe. Nous distinguons :

— Les méthodes d'échantillonnage,

— L'analyse des composantes géantes,

— La détection de clusters,

— L'approximation de clusters.

4.1 Échantillonnage d'arêtes dans un flux

Un panorama des différentes techniques d'échantillonnage sur les graphes est présenté dans [38]. Nous considérons seulement l'échantillonnage d'arêtes [1] dans un flux d'arêtes :

— Échantillonnage uniforme par réservoir,

— Échantillonnage biaisé.

Les problèmes sous-jacents liés à l'échantillonnage de flux de données nécessitent l'application de nombreuses idées et techniques issues de l'échantillonnage de base des données traditionnelles, mais ils nécessitent également de nouvelles innovations significatives et, en particulier pour gérer les requêtes sur des flux de longueur infinie. En effet, la nature illimitée des données en continu représente un écart important par rapport au contexte traditionnel.

Notre discussion porte sur le problème de l'obtention d'un échantillon à partir d'un flux d'arêtes, où la taille d'échantillon souhaitée est beaucoup plus petite que le nombre d'élément dans le flux. Nous faisons une distinction importante entre une fenêtre fixe, dont les points d'extrémités sont des moments spécifiés ou des positions spécifiées dans la séquence du flux, et une fenêtre glissante dont les points d'extrémités avancent au fur et à mesure que le temps avance. Les exemples de ce dernier type de fenêtre comprennent « les éléments les plus récents dans le flux » et « les éléments arrivés au cours de la dernière heure ». L'échantillonnage d'un flux fini est un cas particulier d'échantillonnage à partir d'une fenêtre fixe, correspondant aux premiers et derniers éléments du flux.

Lorsqu'on traite d'une fenêtre stationnaire, de nombreux outils et techniques traditionnels d'échantillonnage de bases de données peuvent être directement utilisés. En général, l'échantillonnage à partir d'une fenêtre glissante est un problème beaucoup plus difficile que l'échantillonnage d'une fenêtre fixe : dans le premier cas, les éléments doivent être retirés de l'échantillon à leur expiration, mais le maintien d'un échantillon de taille adéquate

1. Notons que dans [43], d'autres approches d'échantillonnage sont développées à partir des nœuds ainsi que sur l'exploration.

peut être complexe, ce qui rend la garantie de l'uniformité difficile en ce qui concerne une fenêtre en mouvement.

En effet, dès lors que l'on supprime des éléments expirés [2] dans la fenêtre, on voit apparaître certains cas où l'on ne peut pas garantir l'uniformité, bien qu'il existe quelques méthodes pour s'en approcher.

Les algorithmes que nous avons étudiés et qui reposent sur un réservoir, tombent dans la catégorie d'échantillonnage à probabilité égale, puisque chaque élément de la fenêtre est également susceptible d'être inclus dans l'échantillon.

Pour certaines applications, il peut être souhaitable de biaiser un échantillon vers des éléments plus récents. Dans les sections suivantes, nous allons discuter des schémas d'échantillonnage à probabilité égale et biaisée. Nous considérons une fenêtre stationnaire contenant n éléments e_1, e_2, ..., e_n, énumérés dans l'ordre d'apparition. Si les extrémités de la fenêtre sont définies en termes de points temporels t_1 et t_2, alors le nombre n d'éléments dans la fenêtre est éventuellement aléatoire ; ce fait n'affecte pas matériellement notre discussion, à condition que n soit suffisamment grand pour que l'échantillonnage de la fenêtre en vaille la peine.

4.1.1 Échantillonnage par réservoir

Le réservoir sampling est un algorithme probabiliste qui reproduit un tirage sans remise, il fait partie de la famille des algorithmes randomisés, et a été introduit par Jeffrey Vitter [59]. Soit R un réservoir contenant au plus k arêtes dans un flux de m arêtes $e_1, e_2, ...e_m$. Nous remplissons d'abord le réservoir avec $e_1, e_2, ...e_k$. Pour $i > k$, on décide de garder e_i avec probabilité k/i et si on garde e_i, on enlève une des arêtes du réservoir (avec probabilité $1/k$) pour faire de la place à e_i. L'espace probabiliste Ω est déterminé par les choix effectués à chaque étape par l'échantillonnage du réservoir.

Il faut montrer que chaque arête e_i a alors une probabilité de $\frac{k}{m}$ d'être dans le réservoir, c'est-à-dire uniforme. On procède par récurrence. Le résultat est vrai pour $k = m$. La probabilité qu'un élément de l'ensemble appartienne à l'échantillon est $\frac{k}{m-1}$. À l'ordre m, il y a une probabilité $\frac{k}{m}$ de remplacer un des éléments de l'échantillon. L'élément m a donc une probabilité $\frac{k}{m}$ de faire partie de l'échantillon. Pour les éléments qui en font déjà partie, leur probabilité de rester est :

$$\frac{m-k}{m} + \frac{k}{m}\frac{k-1}{k} = \frac{m-1}{m}.$$

2. Un élément est définie comme expiré si son horodatage ne fait plus partie de la fenêtre

La probabilité qu'un élément présent dans l'échantillon reste dans l'échantillon à l'itération m est de :

$$\frac{m-1}{m}\frac{k}{m-1} = \frac{k}{m}.$$

L'échantillon produit par le réservoir sampling a les mêmes propriétés qu'un échantillon produit avec un tirage sans remise. Cet algorithme est intéressant lorsqu'on veut échantillonner sur une grande base de données car il nécessite de ne garder en mémoire que l'échantillon. Il est également très intéressant de l'exploiter sur des flux de données tels que des flux d'arêtes. En effet, notre expérimentation montre qu'un tel réservoir permet de générer un sous-graphe de taille fixe uniforme.

Algorithme 1 Approche classique du reservoir sampling

Require: $|k|$ // size of a reservoir k
  $m = 0$
  for each edge arriving from the input stream **do**
    $m = m+1$
    if $m \leq |k|$ **then**
      add the edge to the reservoir
    else
      decide with the probability $\frac{k}{m}$ whether to accept the edge
      if the edge is accepted **then**
        replace a randomly selected edge in the reservoir with the accepted edge
      end if
    end if
  end for

```
sample = []
edge = (origine, destination)

def reservoir_sampling(edge, k):
        for index, edge in enumerate(edge_list):
                if index < k:
                        sample.append(edge)
                else:
                        j = random.randint(0, index)
                        if j < k:
                                sample[j] = edge
    return sample
```

Code 4.1: Exemple d'implémentation du réservoir sampling en Python

4.1.2 Echantillonnage biaisé

En général, les échantillons biaisés reposent sur une fonction de biais temporel, afin de favoriser les données récentes dans le flux. Le réservoir biaisé [1] présenté par Aggarwal fournit un biais exponentiel en faveur des éléments récemment observés.
La fonction de biais est donnée par $f(r,t)$ où r est l'élément et t son horodatage.

$$f(r,t) = e^{-\lambda(t-r)} \tag{4.2}$$

Le paramètre λ définit le taux du biais, il est généralement compris dans l'intervalle $[0,1]$

D'autres approches peuvent être utilisées [48] afin d'éviter qu'il y ait une surreprésentation de certains mots-clés dans le flux d'arêtes. Nous pouvons supprimer toutes les arêtes contenant en son extrémité, un de ses mots-clés. Par exemple, la probabilité qu'une arête soit acceptée si son extrémité contient un de ses mots-clés est $p = 0$, tandis que si une arête ne contient pas un mot-clé en son extrémité sa probabilité d'être prise est $p = \frac{k}{m}$.

4.2 Composantes géantes des réservoirs

Le modèle classique d'Erdös-Renyi échantillonne toutes les arêtes avec la même probabilité, en partant du graphe complet. Dans ce cas la probabilité de transition de phase, c'est-à-dire la probabilité d'observer une transition de phase est de $p = 1/n$. Quelle est la transition de phase si on part d'une γ-clique et non pas d'une clique ?

De nombreux travaux sur les graphes aléatoires étudient des conditions suffisantes pour préciser la transition de phase. En particulier, le théorème 1 où d_u est le degré d'un nœud u.

Théorème 1 ([20]). *Soit* $\bar{d} = \frac{\sum_u d_u^2}{\sum_u d_u}$. *Soit* $\epsilon > 0$ *fixé et* $\widehat{G}$ *le sous-graphe aléatoire de* G *obtenu en choisissant chaque arête avec probabilité* p. *Si* $p > (1+\epsilon)/\bar{d}$ *alors* $\widehat{G}$ *a une composante géante.*

Une conséquence directe est le lemme suivant :

Lemme 1. *Soit* $\epsilon > 0$ *fixé. Si* S *est une* γ*-clique et* $p > \frac{1+\epsilon}{\gamma.|S|}$ *alors le sous-graphe aléatoire* $\widehat{S}$ *obtenu dans* $G(|S|,p)$, *a une composante géante.*

Démonstration : Remarquez que $\bar{d} = \frac{\sum_u d_u^2}{\sum_u d_u} \geq \frac{\sum_u \gamma^2.|S|^2}{\sum_u d_u} \geq \frac{\gamma^2.|S|^3}{\gamma.|S|^2} = \gamma.|S|$. Dans la première inégalité. $\sum_u d_u^2$ est minimisé quand les degrés sont uniformes. La deuxième inégalité utilise la définition d'une γ-clique : $\sum_u d_u \geq \gamma.|S|^2$. D'où :

$$\frac{1}{\bar{d}} \leq \frac{1}{\gamma.|S|} \tag{4.3}$$

Nous concluons que si $p > \frac{1+\epsilon}{\gamma.|S|} \geq \frac{1+\epsilon}{d}$ et par le Théorème 1, $\widehat{S}$ a une composante géante.

Ce résultat est intuitif. En passant d'une clique à une γ-clique, la transition de phase est divisée par γ. On va utiliser ce résultat pour quantifier l'existence d'une composante géante dans le réservoir de taille k.

4.3 Méthodes de détection de clusters dans un graphe social

Nous considérons le problème de décision : *existe-t-il un γ-cluster de taille supérieure à $\delta.\sqrt{n}$ dans une graphe social, c'est-à-dire un graphe qui a m arêtes où $m = O(c.n.\log(n))$?* Nous étudions ensuite l'approximation de clusters et calculons des sous-ensembles $C_1, ...C_m$. Dans le cas d'un seul cluster C_1 calculé par l'algorithme, on dit que C_1 ϵ-approxime un cluster C si la similarité de Jaccard $J(C, C_1) \geq 1-\epsilon$ est grande ou si la distance de Jaccard $dist(C, C_1) \leq \epsilon$ est petite.

4.3.1 Graphes qui admettent un grand γ-cluster

Considérons une clique S de taille n' dans le graphe : son image dans un réservoir de taille k est l'ensemble G_S des arêtes $e = (u, v)$ dans le réservoir, où $u, v \in S$. Chaque arête de la clique S est sélectionnée avec une probabilité constante k/m, donc nous sommes dans le cas du modèle Erdös-Renyi $G(n', p)$ où $n' = |S|$ et $p = k/m$.

Nous savons que la transition de phase se produit à $p = 1/n'$, c'est-à-dire qu'il y a une composante géante si $p > 1/n'$ et le graphe est connecté si $p' > \log n/n$. Dans le cas d'un γ-cluster S la transition de phase se fait à $p = 1/\gamma.n'$ d'après le lemme 1. Soit V_S l'ensemble des nœuds de la composante géante G_S dont les nœuds sont dans S. Pour décider de la propriété du graphe : *il y a un grand γ-cluster de taille supérieure à $\delta.\sqrt{n}$* , nous proposons l'algorithme simple A_1, qui est aussi implicite dans [32, 46] mais en stockant $O(n)$ arêtes.

Algorithme $A_1(x, \gamma, \delta)$ pour la détection statique de cluster : Soit un réservoir de taille $k = \frac{c\sqrt{n}\log n}{\gamma.\delta}$. Soit C la composante géante du réservoir. Si $|C| \geq n^{1/8}\log^2 n$, A_1 Accepte, sinon A_1 Refuse.

Lemme 2. *Pour un graphe G dont le degré suit une loi de puissance, s'il existe un γ-cluster S et si $|S| \geq \delta.\sqrt{n}$, alors $Prob_\Omega[|V_S| > n^{1/8}\log^2 n] \geq 1 - \epsilon$.*

Démonstration : Si S est un γ-cluster, alors la transition de phase se produit à $p = 1/\gamma.|S|$.

$$p = \frac{k}{m} = \frac{c\sqrt{n}\log n}{\gamma.\delta.cn.\log n} = \frac{1}{\gamma.\delta.\sqrt{n}} \geq \frac{1}{\gamma.|S|} \tag{4.4}$$

La dernière inégalité vient de l'hypothèse : $|S| \geq \delta.\sqrt{n}$. Donc il y a une composante géante dans le réservoir. La taille de C est donc $O(\sqrt{n})$ et donc supérieure à $n^{1/8}\log^2 n$ avec haute probabilité $1 - \epsilon$.

On souhaiterait aussi montrer, que si G n'a pas un γ-cluster tel que $|S| > \delta.\sqrt{n}$, alors il n'y a pas de composante géante. Nous devons cependant relâcher cette condition et ne considérer que les graphes aléatoires qui satisfont une distribution de degré sous forme de loi de puissance.

Théorème 2. *Soit $\alpha = c.n^{1/8}$. Si S est un γ-cluster et $|S| \geq \delta.\sqrt{n}$ alors :*

$$Prob_\Omega[\text{Algorithme 1 Accepte}] \geq 1 - \epsilon$$

Pour un graphe aléatoire qui satisfait une distribution de degré sous forme de loi de Zipf :

$$Prob_\Omega[\text{Algorithme 1 Refuse}] \geq 1 - \epsilon$$

Démonstration : Soit $|S| \geq \delta.\sqrt{n}$, le Lemme 2 précédent stipule que $Prob_\Omega[\text{Algorithme 1 Accepte}] \geq 1 - \epsilon$. Pour un graphe aléatoire qui satisfait une distribution de degré sous forme de loi de Zipf, [49] montre que la plus grande composante connectée a une taille $O(n^{1/8})$. Par conséquent $Prob_\Omega[\text{Algorithme 1 Rejette}] \geq 1 - \epsilon$.

Notez que dans le cas de graphes aléatoires concentrés, on peut trouver un S de taille $\delta.\sqrt{n}$ en prenant les noeuds de plus grand degré.

4.3.2 Détection de clusters dans un flux d'arêtes de graphes aléatoires

Considérons un flux d'arêtes d'un graphe qui suit une distribution de degrés de loi de puissance. Nous sélectionnons chaque arête avec une probabilité uniforme. À tout moment, nous avons un réservoir qui garde les k arêtes parmi les m arêtes possibles qui ont été lues ($m >> k$), et chaque arête a la même probabilité $\frac{k}{m}$ d'être choisie dans le réservoir.

La méthode d'échantillonnage que nous proposons n'est pas nouvelle, c'est une des méthodes d'échantillonnage de [5, 38]. Si un cluster est grand, il y aura une grande composante connectée dans le réservoir à titre de témoin. Nous ignorons toutes les petites composantes connectées du réservoir et ne stockons dans une base de données que les grandes composantes.

Dans une expérience typique $k = 400$ alors qu'on lit 10^4 arêtes dans une fenêtre. Nous ignorons toutes les composantes de taille inférieure à un certain seuil, par exemple à 10. Pour un graphe G, il peut y avoir plusieurs grands clusters C_j ou bien, il peut n'y en avoir aucun.

4.3.2.1 Cas uniforme

Le modèle de configuration [53] fixe une distribution arbitraire des degrés des nœuds du graphe, $\mathcal{D} = [D(1), D(2), ...D(k)]$ où $D(i)$ est le nombre de nœuds du degré i. Il génère un graphe aléatoire parmi tous les graphes avec n nœuds, où $n = \sum_i D(i)$ et $\sum_i i * D(i)/2$ arêtes. Par exemple si $\delta = [4, 3, 2]$ [3], c'est-à-dire approximativement la loi de puissance, nous cherchons un graphe avec 9 nœuds et 8 arêtes. Spécifiquement 4 nœuds de degré 1, 3 nœuds de degré 2 et 2 nœuds de degré 3, comme dans la Figure 4.1. (a). Alternativement, nous pouvons représenter $\mathcal{D}$ comme une distribution, c'est-à-dire $\mathcal{D} = [\frac{4}{9}, \frac{1}{3}, \frac{2}{9}]$.

Le modèle de configuration génère des graphes avec la distribution des degrés $\mathcal{D}$ lorsque $\sum_i i * D(i)$ est pair. On énumère les demi-arêtes (stubs) des nœuds en fonction de leurs degrés, et on sélectionne une permutation (matching) symétrique aléatoire π entre les demi-arêtes de sorte que $\pi(i) \neq i$. Le graphe peut avoir des boucles et des arêtes multiples, si on veut un graphe simple il suffit de répéter la procédure [35]. Si $\mathcal{D}$ suit une loi de puissance de Zipf, alors :

$$Prob[d(u) = i] = c/i^2 \tag{4.5}$$

Le degré moyen est $c. \log n$ et le degré maximum est $\sqrt{c.n}$ si le graphe a n nœuds et $m = c.n.logn$ arêtes. Si la permutation π est sélectionnée avec la distribution uniforme,

3. Alternativement, on peut donner une séquence d'entiers, les degrés des différents nœuds dans l'ordre croissant, c'est-à-dire $[1, 1, 1, 1, 2, 2, 2, 3, 3]$, une séquence de longueur 9 pour la distribution $\delta = [4, 3, 2]$.

nous disons que le graphe G est choisi par la sélection uniforme.

Un graphe typique généré par la sélection uniforme n'a pas de grand cluster. Nous étudions une variante où la permutation π n'est pas choisie uniformément mais suit une concentration sur un sous-ensemble S de nœuds.

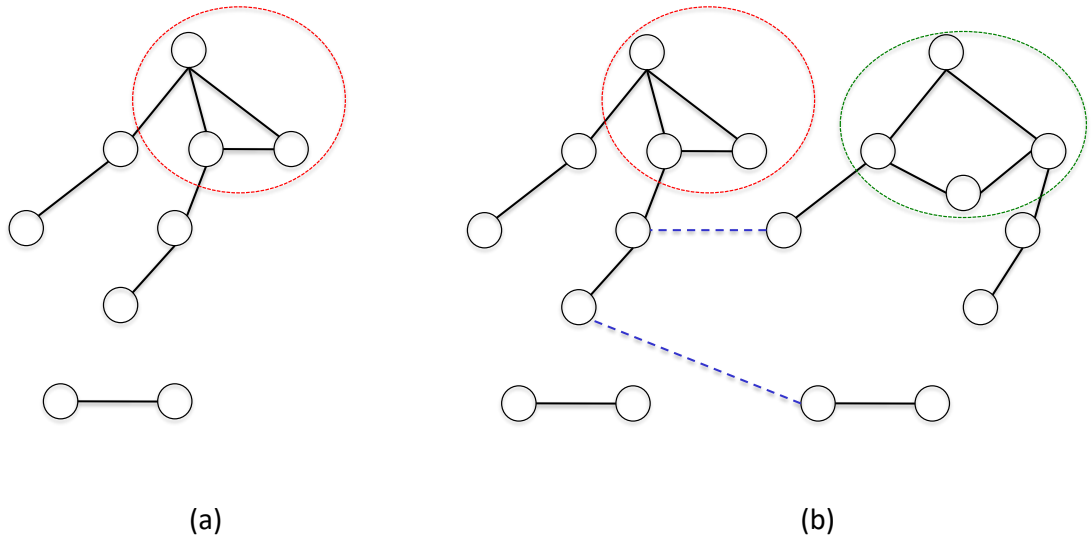

FIGURE 4.1: Graphe aléatoire concentré pour D avec une communauté dans (a). Graphe aléatoire proche de D avec 2 communautés dans (b) où $\delta = [4, 3, 2]$ (ou $[\frac{4}{9}, \frac{1}{3}, \frac{2}{9}]$ comme distribution)

4.3.2.2 S-Concentration

Fixons un sous-ensemble S parmi les nœuds de haut degré et choisissons la permutation π comme suit :

— Un stub dans S choisi avec probabilité γ de définir une arête dans S, pour cela il choisit une demi-arête libre. On choisit ainsi de faire correspondre les demi-arêtes de S uniformément dans S,

— Avec probabilité $1 - \gamma$ faire correspondre les demi-arêtes de S uniformément dans $V - S$. Faire correspondre uniformément les demi-arêtes restantes.

Dans le cas d'un graphe aléatoire choisi par S-concentration, on observe un γ-cluster avec grande probabilité et si S est suffisamment grand, le réservoir le détectera d'après le théorème 2.

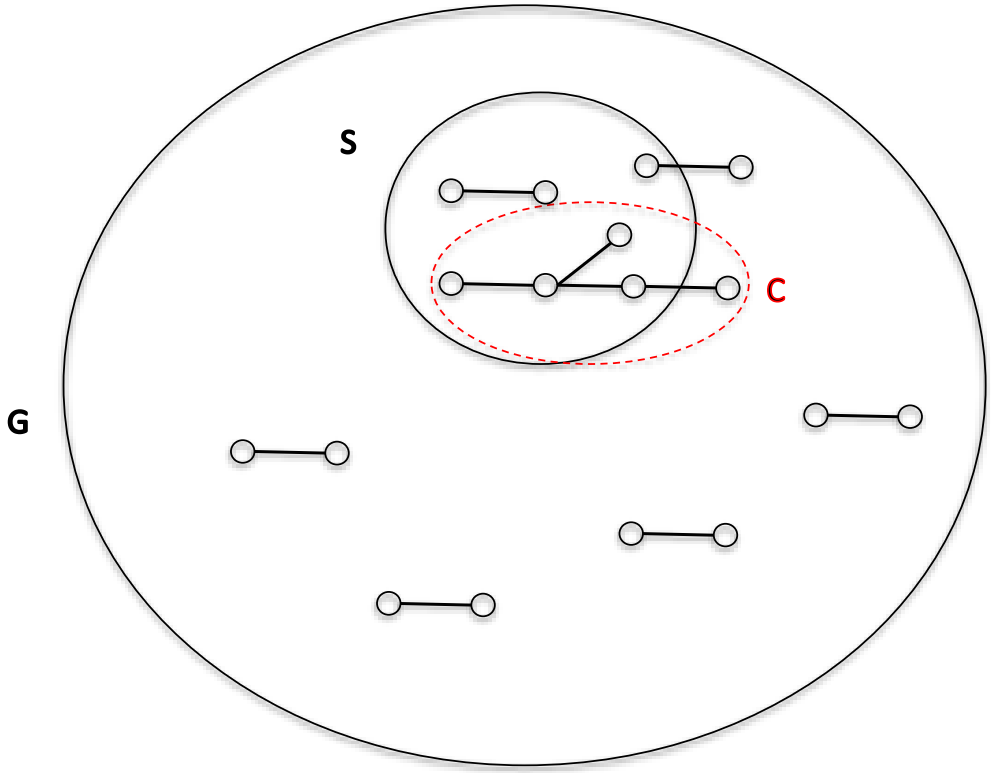

FIGURE 4.2: Graphe aléatoire concentré G avec une communauté S et 10 arêtes aléatoires du réservoir définissant $\widehat{G}$ avec la grande composante connectée C avec 4 arêtes

4.4 Approximation de clusters

On peut aussi s'intéresser à l'approximation des clusters. Une technique classique de théorie des graphes est d'introduire le 2-core d'un graphe connexe. C'est le graphe obtenu en éliminant de façon itérative tous les nœuds de degré 1. On peut généraliser cette notion au d-core pour $d > 2$.

Pour un graphe S-concentré et S suffisamment grand, le 2-core de la composante géante du réservoir est une bonne approximation de S. Soit l'algorithme A qui suit l'algorithme A_1 et dans le cas où il y a une composante géante H, calcule 2-core $(H) = H_1$. on peut montrer que :

$$Prob_{\Omega}[dist(A(x), S) \leq \epsilon] \geq 1 - \delta \tag{4.6}$$

La notion de d-core est aussi introduite dans ce cadre dans [12].

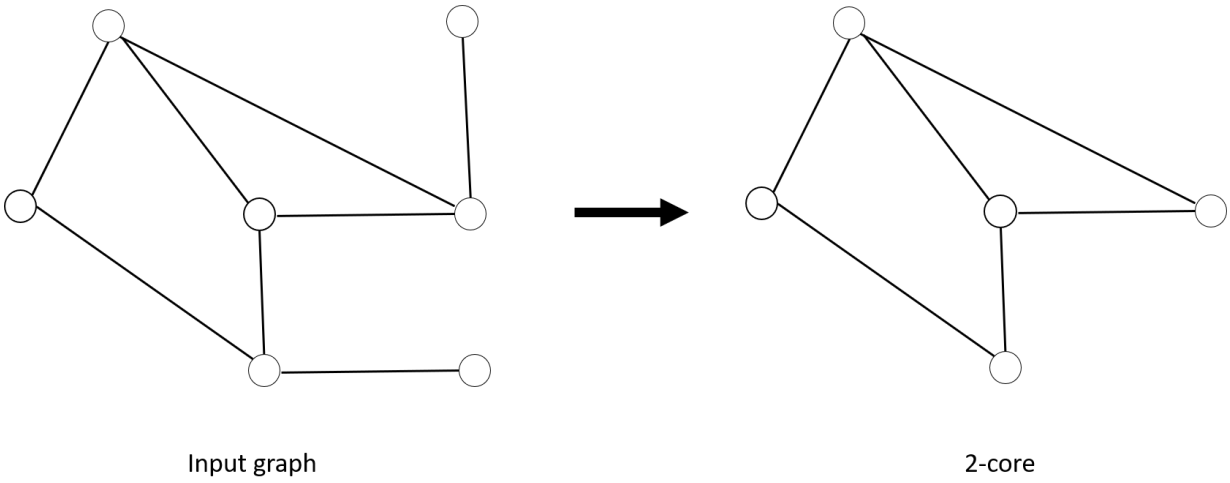

FIGURE 4.3: Illustration du 2-core pour un graphe G .

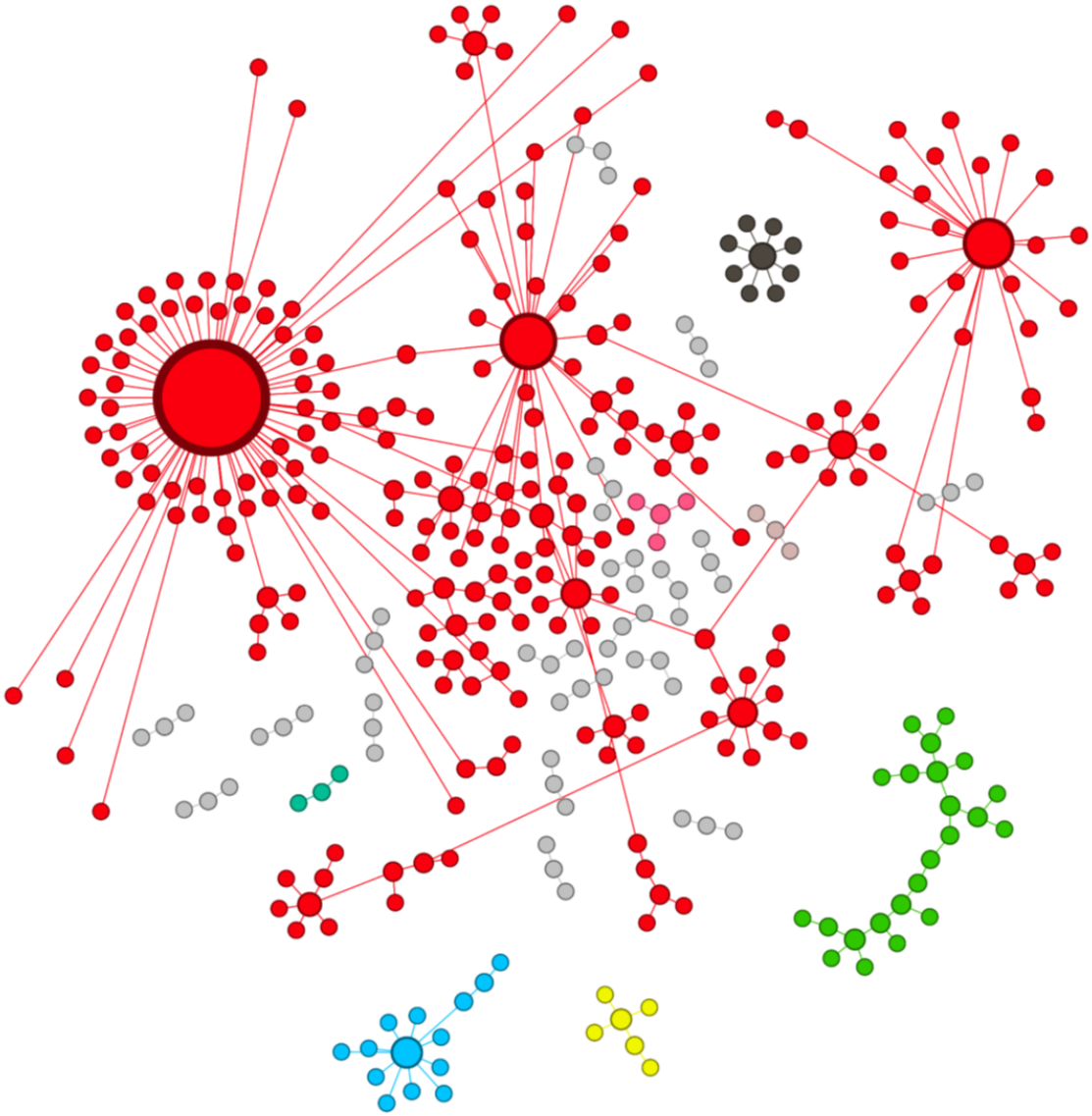

FIGURE 4.4: Composantes connexes dans un k-réservoir.

4.5 Contributions dans le domaine

Dans cette partie, il s'est agi de montrer comment nous avons réussi à franchir la barrière de $O(n)$ présente dans l'article [8], en utilisant une hypothèse sur les graphes sociaux. Cette hypothèse basée sur la distribution des degrés qui suit une loi de puissance nous a permis de passer à une mémoire sous-linéaire, soit $O(\sqrt{n}.\log(n))$. Nous avons également montré qu'avec cette mémoire nous pouvons garantir, avec une grande probabilité, la détection d'un gros cluster de taille $O(\sqrt{n})$ dans un graphe social statique défini par un flux d'arêtes, sans stocker l'ensemble du graphe. Si nous avons un graphe aléatoire qui n'a pas de cluster, l'algorithme va le détecter et sa réponse sera correcte. Nous pouvons également approximer le cluster du graphe à partir du 2-*core* de la composante géante du réservoir.

5 Méthodes d'approximation de clusters sur un graphe social dynamique

Nous limitons maintenant notre attention aux flux de données infinis et considérons les méthodes d'échantillonnage à partir d'une fenêtre glissante contenant les éléments de données les plus récents. Comme il a été mentionné précédemment, cette tâche est sensiblement plus difficile que l'échantillonnage d'une fenêtre stationnaire. La difficulté vient du fait que les éléments doivent être retirés de l'échantillon lorsqu'ils expirent, de telle sorte que le maintien d'un échantillon soit uniforme pour chaque fenêtre.

Fenêtre dynamique. Nous distinguons les fenêtres basées sur les séquences et les fenêtres basées sur l'horodatage. Une fenêtre basée sur une séquence de longueur n contient les n éléments les plus récents, tandis qu'une fenêtre de longueur t basée sur un horodatage contient tous les éléments arrivés au cours des t unités de temps précédentes.

Parce qu'une fenêtre glissante favorise intrinsèquement les éléments récemment arrivés, nous nous concentrons sur des techniques d'échantillonnage à probabilité égale à l'intérieur de la fenêtre elle-même.

Dans de nombreuses applications, les données considérées comme importantes (ou intéressantes) sont les données les plus récentes. Tandis que les données anciennes sont considérées comme obsolètes et ne sont plus utilisées lors de l'évaluation des requêtes.

Nous considérons le problème du maintien d'un échantillon aléatoire uniforme d'une taille spécifiée k sur une « fenêtre en mouvement », comme des éléments les plus récents d'un flux de données. Nous présentons un algorithme efficace pour ce problème sous la définition d'une fenêtre en mouvement. Une fenêtre de durée t est constituée de tous les éléments de données dont l'horodatage d'arrivée se trouve dans un intervalle t de l'heure actuelle.

Le problème est de savoir comment maintenir un échantillon d'une taille spécifiée sur des données arrivant en ligne. La solution standard consiste à utiliser les techniques d'échantillonnage de réservoir de Vitter.

L'échantillonnage des réservoirs fonctionne bien lorsque les données à venir ne contiennent que des insertions, mais il rencontre des difficultés si les données contiennent des suppressions, comme c'est le cas lorsque les données expirent. Une solution a été présentée dans [17] en considérant deux fenêtres disjointes et en interpolant les échantillons pour toutes fenêtres intermédiaires. Cette technique induit un délai et nous proposons une technique sans délai.

Dans ce chapitre nous allons étudier les méthodes d'échantillonnage par réservoir sur un graphe dynamique. Nous introduisons notre algorithme, le step continous réservoir qui repose sur une approche hybride, regroupant différentes méthodes d'échantillonnage sur des fenêtres en mouvement.

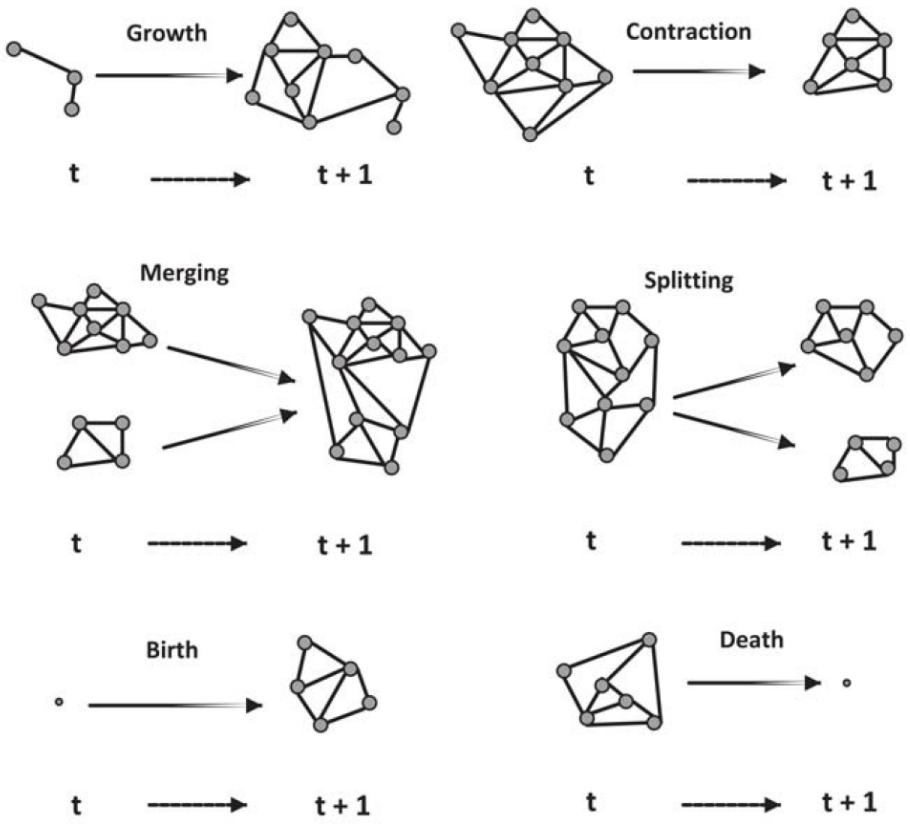

FIGURE 5.1: Exemple d'évolution de cluster [54] dans un graphe dynamique

5.1 Échantillonnage à partir d'un réservoir dynamique

De notre point de vue, les données générées sur les réseaux sociaux tels que Twitter, nécessitent une approche par approximation lorsque nous souhaitons les suivre sous l'angle de graphes dynamiques, c'est-à-dire un graphe qui évolue au cours du temps. Nous supposons ici que les arêtes et les noeuds évoluent, ce qui induit que l'ensemble des sommets sont instables.

5.1.1 L'échantillonnage stratifié

L'échantillonnage stratifié permet d'obtenir la fenêtre divisée en segments disjoints de même longueur. Le schéma d'échantillonnage stratifié pour une fenêtre statique peut être adapté pour obtenir un échantillon stratifié et un réservoir dynamique.

Le schéma le plus simple consiste à diviser le flux en strates de longueur λ, où λ divise la longueur de la fenêtre τ ; voir Figure 5.2. L'échantillonnage par réservoir est utilisé ici pour obtenir un échantillon aléatoire simple de taille k pour chaque strate. Les éléments échantillonnés expirent de la manière habituelle. La fenêtre actuelle contient toujours l strates, où $l = \frac{\tau}{\lambda}$, et toutes sauf peut-être la dernière strate sont de la même longueur, à savoir λ.

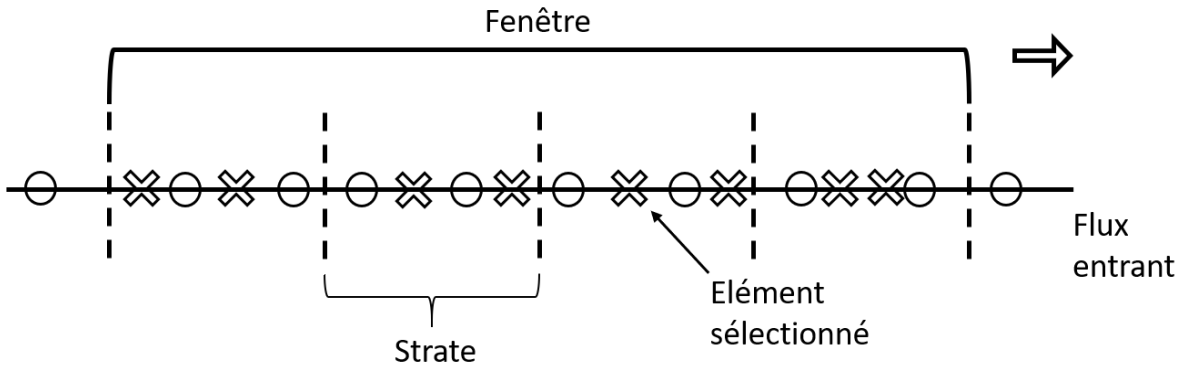

FIGURE 5.2: Échantillonnage stratifié pour une fenêtre glissante

5.1.2 Step reservoir sampling

Nous présentons ci-dessous notre algorithme qui a pour objectif de répondre aux contraintes imposées par le flux d'arêtes issu de Twitter, lequel se positionne comme un algorithme d'échantillonnage basé sur des fenêtres avec un horodatage. Nous nous sommes inspirés du réservoir de Vitter adapté à une fenêtre stationnaire et nous l'avons adapté sur une fenêtre en mouvement, de flux variable en combinant plusieurs méthodes vues précédemment.

Supposons un flux d'arêtes avec un horodatage $e_1, e_2, ...e_i...$ où chaque $e_i = (u, v, t)$. Dans notre modèle de fenêtre dynamique nous isolons les arêtes les plus récentes à chaque temps discrets tels que $t_1, t_2, ...$ Nous fixons la longueur τ d'une fenêtre, donc $t_1 = \tau$ et chaque $t_i = \tau + \lambda.(i-1)$ pour $i > 1$ et $\lambda < \tau$ détermine une fenêtre w_i de longueur τ.

Nous définissons dans notre algorithme plusieurs paramètres, tels que :

— k définit le nombre maximal d'arêtes que l'on souhaite avoir dans le réservoir.

— τ définit la longueur de la fenêtre souhaitée de notre réservoir.

— λ définit le pas de temps souhaité dans la fenêtre.

Nous avons par ailleurs des variables qui sont dépendantes des données telles que :

— M est une liste de longueur $\frac{\tau}{\lambda}$, qui représente le nombre de steps dans une fenêtre w_i. Nous incrémentons chaque élément de la liste en fonction du nombre d'arêtes vues dans la step.

— m_i nombre de tuples dans la fenêtre w_i, est égale à la somme des M_J

— k_d définit le nombre d'arêtes dans le réservoir.

Notre algorithme gère le réservoir dynamique et les fenêtres comme suit :

Gestion des fenêtres dynamiques. Supposons que nous sommes à la fenêtre w_i dans l'intervalle de temps $[t_i - \tau, t_i]$, une arête e_j arrive avec un horodatage :

— Si son horodatage est compris dans l'intervalle $[t_i - \tau, t_i]$ nous appliquons la gestion du réservoir présentée ci-dessous.

— Si son horodatage n'est pas compris dans l'intervalle $[t_i - \tau, t_i]$:

1. Nous définissons une nouvelle fenêtre w_{i+1} qui est égale à la fenêtre $w_i + \lambda$
2. Nous supprimons du réservoir les arêtes jugées comme étant périmées, c'est-à-dire que leur horodatage ne fait plus partie de l'intervalle de la fenêtre w_{i+1}, et nous mettons à jour k_d
3. Nous mettons à jour m_i, pour cela, nous décalons les éléments de la liste M, ceci nous permet de maintenir en permanence l'ensemble des arêtes vues dans chacune des steps. En faisant la somme sur les différentes steps nous pouvons connaître le nombre d'arêtes vues dans la fenêtre w_i.

4. Nous appliquons à l'arête e_j la gestion du réservoir présentée ci-dessous.

Gestion du réservoir dans la fenêtre w_i. Un flux d'arêtes avec un horodatage $e_1, e_2, ...e_i...$ où chaque $e_i = (u, v, t)$ remplit notre *réservoir* tant que le nombre de tuples $k_d \leq k$ à l'étape courante.

Les arêtes du réservoir sont stockées dans une liste de façon ordonnée (c'est-à-dire par ordre d'arrivée).

Si $m_i > k$, la probabilité du réservoir de Vitter est appliquée ; c'est-à-dire que chacune des arêtes du flux a la probabilité $\frac{k}{m_i}$ de faire partie du réservoir où m_i est le nombre d'arêtes vues dans la fenêtre w_i. Pour cela, nous tirons au hasard un nombre j compris dans l'intervalle $[0 : m_i]$, si $j \in k$ l'arête est acceptée.

— Si $k_d = k$ nous supprimons l'arête qui occupe la position j dans le *réservoir* et nous acceptons la nouvelle arête qui est ajoutée dans le réservoir, dans l'ordre d'arrivée.

— Dans le cas où $k_d < k$, l'arête est ajoutée au début du réservoir sans suppression d'arêtes.

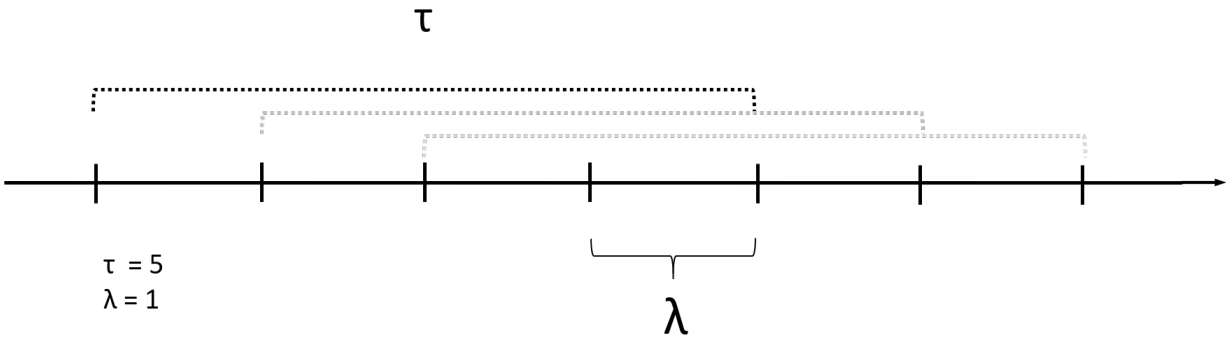

FIGURE 5.3: Step reservoir sampling.

5.2 Méthodes de détection de clusters dans un graphe dynamique

La Logique Temporelle est un cadre pour décider des propriétés temporelles des graphes dynamiques G_t. Soit P une propriété du graphe telle que la Connectivité, ou l'existence d'un γ-cluster de grande taille. Une propriété temporelle typique est $\Diamond\ P$ indiquant qu'il existe un t tel que $G_t \models P$ ou $\Box P$ indiquant que pour tous les t, $G_t \models P$. Soit P la propriété précédente : existe t-il un t pour lequel il y a un grand γ-cluster ? Comment décidons-nous $\Diamond\ P$? Nous allons généraliser l'algorithme A_1 introduit au chapitre 5.

Détection dynamique de cluster, Algorithme $A_2(x, \gamma, \delta)$: Soit un réservoir de taille $k = \frac{c\sqrt{n}\log n}{\gamma.\delta}$. Si $|C_i| \geq n_i^{1/8}\log^2 n_i$, A_2 Accepte, sinon A_2 Refuse.

Nous montrons que si le graphe possède de grands clusters pendant un intervalle Δ, nous le détectons avec grande probabilité et que si un graphe aléatoire qui satisfait la distribution de degré n'a pas de grand cluster, nous pouvons aussi garantir que nous n'en détecterons pas.

5.2.1 Graphes avec des grands clusters

Supposons qu'un graphe dynamique $G(t)$ admette un grand cluster pendant au moins un temps $\Delta > \tau$. L'espace probabiliste Ω_t est maintenant beaucoup plus grand, puisqu'il concerne les tirages de toutes les fenêtres qui précèdent l'instant t. Soit $m(t)$ le nombre d'arêtes dans la fenêtre au temps t. Soit $G(t)$ un graphe défini par un flux d'arêtes suivant une loi de puissance $\mathcal{D}$. Un graphe aléatoire uniforme qui satisfait une distribution de degrés comme loi de puissance est défini dans la section suivante. Nous obtenons une généralisation du théorème 2.

Théorème 3. *Pour un graphe dynamique $G(t)$ qui admet un γ-cluster S de taille $\geq \delta.\sqrt{n}$ pendant un temps $\Delta > \tau$, alors :*

$$Prob_\Omega[A_2 \text{ Accepte}] \geq 1 - \delta^{\Delta/\tau} \tag{5.1}$$

Pour un graphe aléatoire $G(t)$ uniforme qui satisfait une loi de puissance :

$$Prob_\Omega[A_2 \text{ Refuse}] \geq 1 - \delta.(t - t_1)/\lambda \tag{5.2}$$

Démonstration : Pour chaque fenêtre, on peut appliquer le théorème 2 et il y a Δ/τ fenêtres indépendantes. Si $|S| \geq \delta.\sqrt{n}$, la probabilité d'erreur est inférieure à l'erreur faite pour Δ/τ independent windows, qui est $\delta^{\Delta/\tau}$.

Par conséquent, $Prob_\Omega$[Algorithme 2 Accepte] $\geq 1 - \delta^{\Delta/\tau}$.

Pour un graphe aléatoire uniforme, l'algorithme doit être correct à chacune des $(t - t_1)/\tau$ fenêtres. La probabilité globale d'erreur est inférieure à $\delta.(t - t_1)/\lambda$. Par conséquent :

$$Prob_{\Omega}[\text{Algorithme 2 Refuse}] \geq 1 - \delta.(t - t_1)/\lambda$$

La probabilité d'accepter est amplifiée alors que la probabilité de rejeter pour la dynamique uniforme diminue. Une seule erreur génère une erreur globale. Pour t grand la borne $(1 - \delta.(t - t_1)/\tau)$ peut devenir négative. Nous pourrions également estimer Δ avec des techniques similaires.

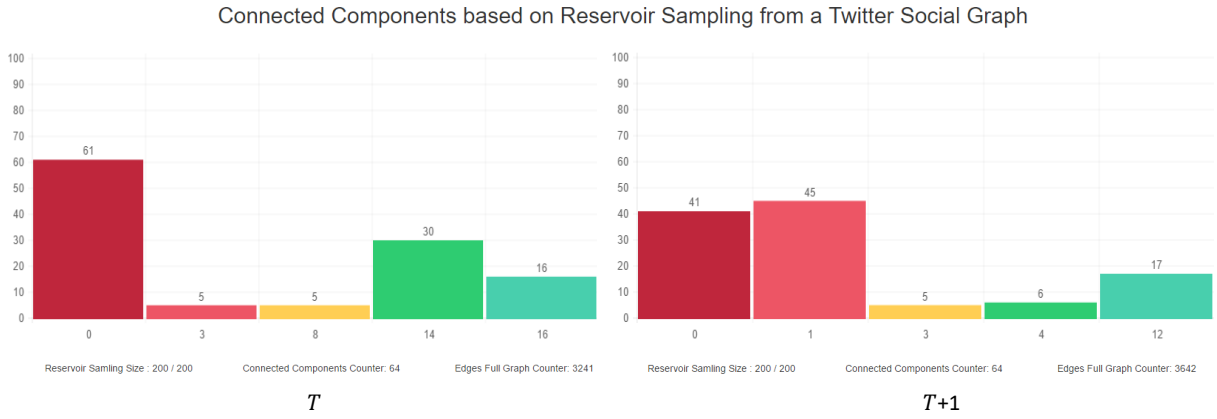

FIGURE 5.4: Taille des composantes connexes en ligne

5.2.2 Graphes aléatoires dynamiques dans un flux d'arêtes

Nous suivons le modèle des fenêtres glissantes défini par un intervalle de temps fixe τ. Si le débit (le nombre d'arêtes par unité de temps) du flux est fixe, chaque fenêtre admet le même nombre d'arêtes. Ce n'est pas le cas en pratique, car le taux fluctue d'un facteur de 2 à tout moment.

Chaque fenêtre de longueur τ se termine aux instants $t_1, t_2, ...t_i, ...$ Chaque $t_i = \tau + \lambda.(i - 1)$ pour $i > 1$ et $\lambda < \tau$ détermine une fenêtre de longueur τ et un graphe G_i défini par les arêtes dans la fenêtre ou dans un intervalle de temps $[t_i\tau, t_i]$. Le nombre d'arêtes dans une fenêtre peut augmenter ou diminuer et reflète la vitesse croissante ou décroissante d'un flux. Les fenêtres consécutives se chevauchent dans un facteur τ/λ, environ 50% dans les expériences. En pratique, $\tau = 60$ minutes et $\lambda = 30$ minutes.

Les graphes G_{i+1} et G_i partagent plusieurs arêtes : les anciennes arêtes de G_i sont supprimées et de nouvelles arêtes sont ajoutées à G_{i+1}. Les graphes sociaux ont une structure spécifique, une distribution de degré spécifique (loi de puissance), un petit diamètre

et quelques clusters denses. Les graphes aléatoires dynamiques présentés dans la section satisfont ces conditions. Nous définissons un modèle de graphe aléatoire dynamique $G(t)$ qui peut ou non avoir des clusters et suivre une distribution de degrés en utilisant le configuration model.

5.2.2.1 Dynamique uniforme

Nous généralisons le configuration model dans un environnement dynamique. En retirant uniformément et de manière aléatoire $q \geq 2$ arêtes sur l'ensemble des arêtes de G, cela libère $2.q$ demi-arêtes (stubs). Pour obtenir G' nous générons un nouveau matching uniforme sur ces hubs. La distribution des graphes aléatoires reste uniforme.

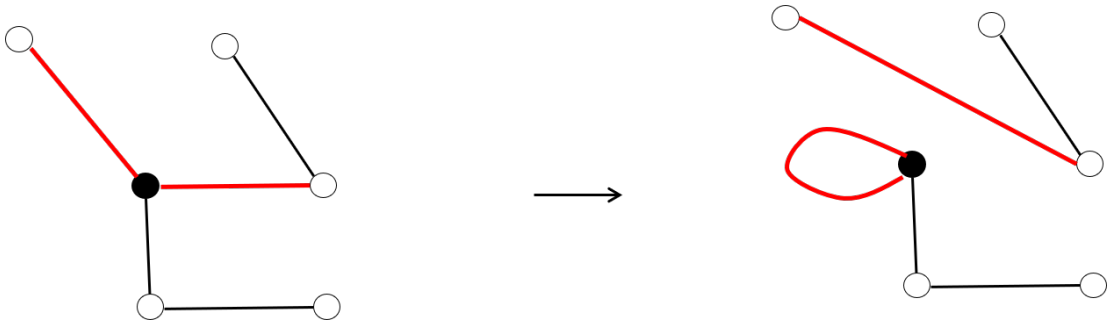

FIGURE 5.5: Exemple dynamique uniforme

5.2.2.2 Dynamique concentrée

Un graphe typique généré par la Dynamique Uniforme n'est pas susceptible d'avoir un grand cluster. **La dynamique S-concentrée** fixe un sous-ensemble S parmi les nœuds de haut degré. Enlever $q \geq 2$ arêtes, uniformément sur l'ensemble des arêtes de G, libérant ainsi $2.q$ demi-arêtes. Une demi-arêtes est dans S si son origine ou son extrémité est dans S. Avec une probabilité γ, faire le matching des demi-arêtes dans S uniformément dans S. Avec une probabilité de $1 - \gamma$, faire le matching des demi-arêtes dans S uniformément dans $V - S$.

Après quelques itérations avec une probabilité élevée, cette dynamique concentrera les arêtes dans S et créera un γ-cluster. En supposant que la distribution des degrés suive une loi de puissance avant la transformation, la distribution des degrés reste identique. Le γ-cluster reste également approximativement identique.

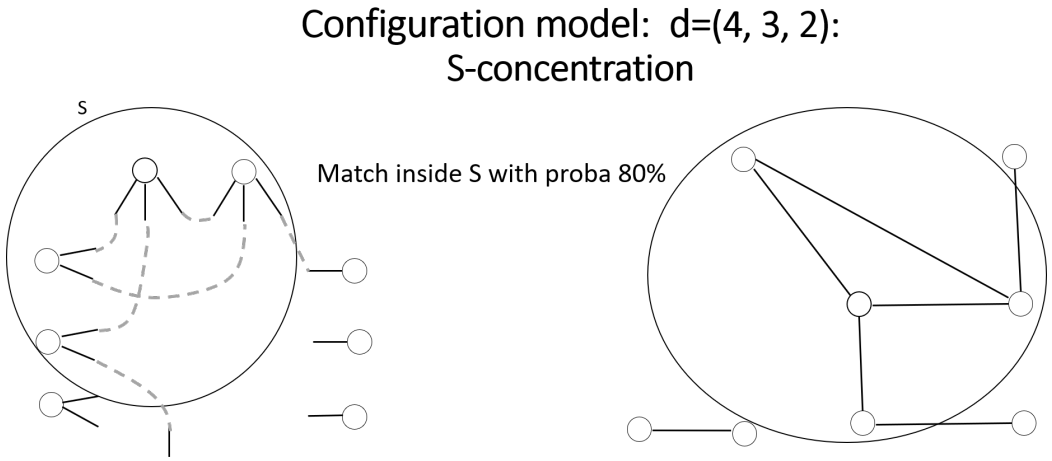

FIGURE 5.6: Exemple de dynamique concentrée

5.2.2.3 Dynamique générale

Une Dynamique générale est une fonction qui choisit à tout moment l'une des deux stratégies suivantes : une Dynamique Uniforme ou une Dynamique concentrée S pour un S fixe. Un exemple est le **Step Dynamics** : appliquez d'abord le Uniform Dynamics, puis passez à S-dynamics pour une période de temps Δ, et revenez aux Uniform Dynamics. Dans notre environnement, la dynamique dépend de certaines informations externes, que nous essayons de récupérer approximativement. Notons que pendant la phase de Dynamique Uniforme, il n'y a pas de grandes composantes et nous ne stockons rien.

Pour la phase Δ, nous stockons que quelques composantes qui se rapprocheront de S. Des stratégies plus complexes pourraient impliquer plusieurs clusters S_1 et S_2 qui peuvent ou non se recouper.

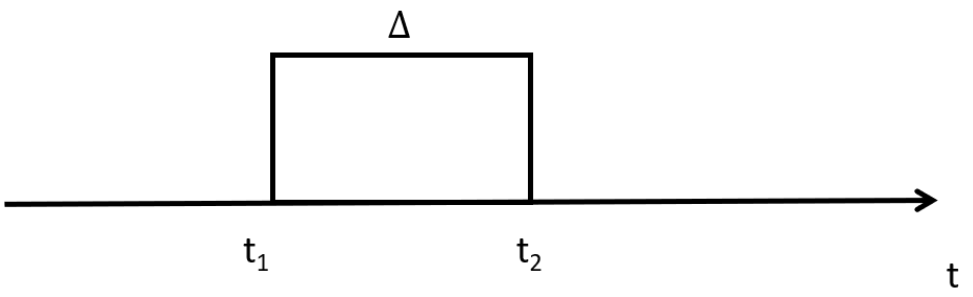

FIGURE 5.7: Exemple de dynamique générale

5.2.3 Stabilité des clusters

En observant la dynamique des communautés, on constate une certaine instabilité : certaines composantes apparaissent, disparaissent et peuvent réapparaître plus tard. La meilleure façon de l'observer est d'utiliser l'expérience suivante : supposons deux réservoirs indépendants de taille $k' = \frac{k}{2}$ comme dans la figure 5.8. Les deux dernières communautés du réservoir 1 et les communautés 5 fusionnent pour correspondre aux communautés 4 du réservoir 2.

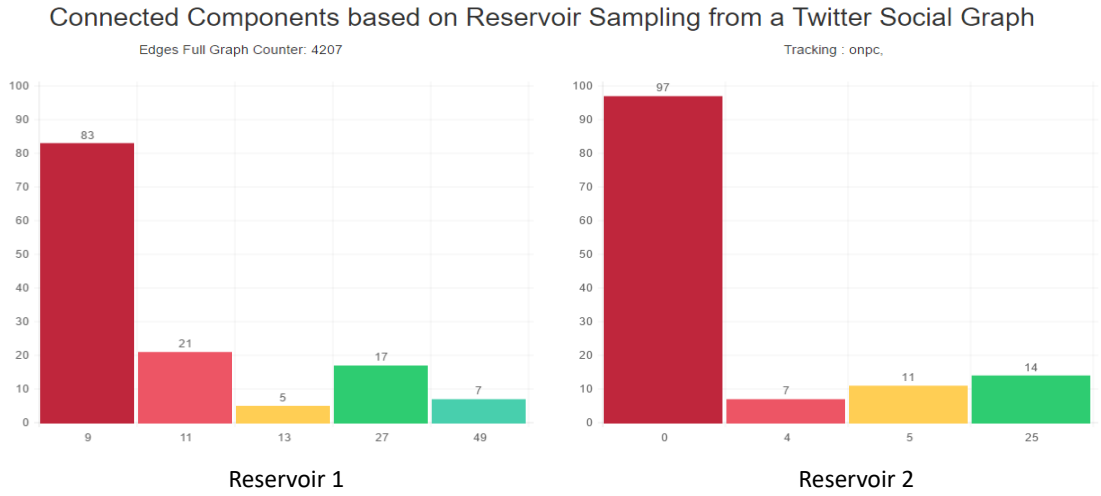

FIGURE 5.8: Taille des composantes connexes dans 2 réservoirs indépendants

Considérons le sous-graphe G_i de la communauté C_i. Il s'agit très probablement d'un arbre si C_i est petit, donc instable car la suppression d'une arête divise la composante ou la rend petite et elle disparaît. Les plus grandes composantes sont des graphes qui sont donc plus stables. Si le graphe original avec m arêtes a une composante concentrée S de taille $O(\sqrt{m/2})$, alors nous pouvons estimer avec le modèle Erdös-Renyi $G(n, p)$ les composantes connectées dans S. Les composantes sont très probablement des arbres de taille maximum $O(\log(\sqrt{m/2})$. D'où l'instabilité des petites composantes.

5.3 Autres approches

Il existe de nombreuses autres approches pour détecter les clusters dans des flux d'arêtes de graphes. La communauté des algorithmes *graphes dynamiques*, par exemple [12] étudie le compromis entre le temps de mise à jour et le temps de requête dans le pire des cas, pour le problème du sous-graphe dense. Nous avons les solutions de [32, 46] qui sont assez proches de la notre.

Le *graph streaming approach* [45] souligne la complexité de l'espace dans le pire des cas et en particulier pour le modèle de fenêtre. L'approche d'échantillonnage du graphe telle que [5] considère l'échantillonnage uniforme sur les arêtes mais il n'y a pas d'analyse pour la détection des clusters dans les graphes dynamiques. La détection d'une clique plantée est un problème classique [33], difficile lorsque la taille de la clique est par exemple $O(\sqrt(n)/2)$ dans le pire des cas. La communauté de graphes mining [4, 2, 63] étudie la détection des clusters lorsque les graphes sont entièrement connus.

Dans notre approche, nous ne considérons que les classes de graphes qui suivent une distribution de degrés de loi de puissance et étudions des algorithmes approximatifs pour la détection de γ-cliques dynamiques dans le modèle de fenêtre en utilisant seulement de petits Réservoirs.

5.4 Contributions dans le domaine

Dans cette partie, nous avons étendu notre approche développée au sein du chapitre précédent au graphe dynamique défini par une fenêtre temporelle. Nous avons montré qu'avec une mémoire sous-linéaire, nous pouvons détecter avec une grande probabilité l'existence d'un gros cluster s'il apparaît pendant un certain temps dans un graphe social dynamique. L'algorithme proposé fait moins d'erreurs pour détecter un grand $\gamma-$cluster dans le cas dynamique que dans le cas statique.

L'étude des graphes sociaux dynamiques nous a conduit à travailler sur une nouvelle méthode d'échantillonnage par réservoir, le "step reservoir sampling" que nous avons présentée. Cette technique nous permet d'approximer en ligne un flux d'arêtes avec une mémoire constante sans délai. Une méthode proche de la notre a été introduite dans [17], cependant elle nécessite d'avoir deux fenêtres contiguës pour générer des échantillons uniformes, ce qui implique un délai.

6 Intégration de données de flux

Traditionnellement, dans le monde des bases de données, l'objectif principal de l'intégration des données est de fournir une vue uniforme, à des sources de données hétérogènes qui suivent des schémas différents. L'intégration des données est de plus en plus fréquente, à mesure que le volume et le besoin de partager les données ne cessent de croître. Comment peut-on répondre à une requête qui nécessite une partie des données de chaque source ?

L'intégration de données dans les bases de données, repose souvent sur l'échange de données. Ces méthodes étudient les schémas de transformation d'un schéma dans un autre, elles ne prennent pas compte des techniques d'approximation. Les algorithmes d'approximation, comme celui que nous proposons, donnent des informations importantes pour l'intégration de sources multiples.

Nous sommes dans le cas où les sources sont des flux de données, plus particulièrement des flux d'arêtes qui partagent une horloge. Prenons deux flux d'arêtes définissant deux graphes $G_i = (V_i, E_i)$ pour $i = 1, 2$, comment combiner ces sources de données dans ce cadre ? Nous allons utiliser la corrélation pour répondre à cette problématique.

6.1 Intégration de flux d'arêtes

Supposons que l'on lise plusieurs flux, comment intégrer les composantes géantes issues des Réservoirs de chacun des flux ?

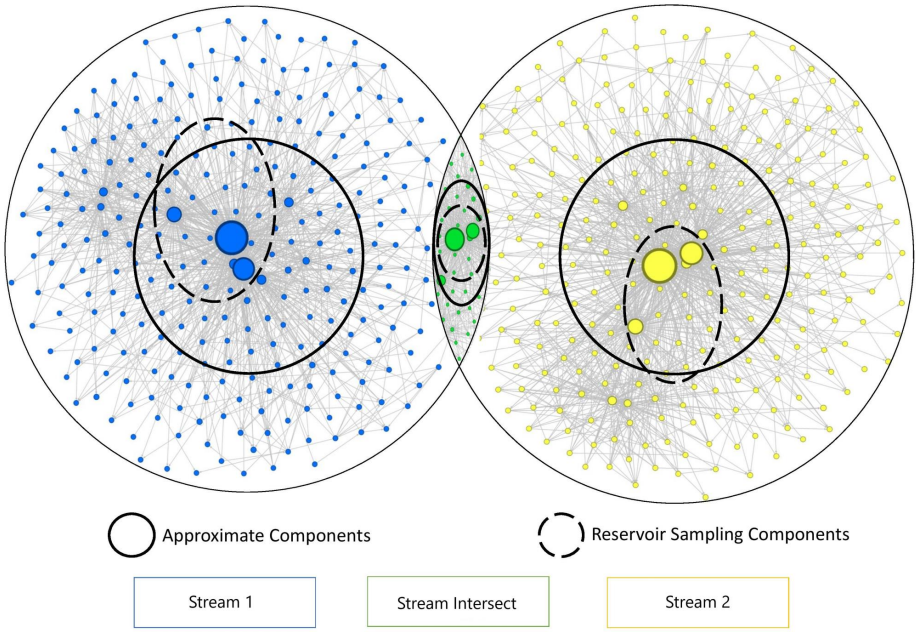

FIGURE 6.1: Communautés communes entre deux graphes

Ce qui est stocké dans le temps. À certains moments discrets $t_1, t_2, ...$, nous stockons les grandes composantes connectées des Réservoirs R_t. Il pourrait n'y en avoir aucune. Nous utilisons une base de données NoSQL, avec 4 tables (Clé, Valeurs) où la clé est toujours un tag (@x ou #y) et les valeurs stockent les nœuds des clusters. Notez qu'un flux est identifié par un tag (ou un ensemble de tags) et qu'un cluster est également identifié par un tag (son noeud le plus haut degré).

— *Stream(tag, list(cluster, timestamp)* est la table qui fournit les clusters les plus récents d'un flux,

— *Cluster(tag, list(stream, timetamp, list(high-degree nodes), list(nodes(nodes,degree))))* est la table qui fournit la liste des nœuds de haut degré et la liste des noeuds avec leur degré, dans un cluster donné,

— *Nodes(tag, list(stream, cluster, timestamp)* est la table qui fournit pour chaque nœud la liste des flux, clusters et timestamps où le nœud apparaît,

— *Corrélation((tag1,tag2), list(value,timestamp))* est la table qui fournit pour chaque paire de flux *(tag1,tag2)* les différentes valeurs de corrélation $\rho(t)$.

6.1.1 Intégration de flux Twitter

Nous avons simultanément capturé 4 flux Twitter [1] sur les tags #CNN, #FoxNews, #Bitcoin, et #Xrp (Ripple) pendant 24 heures avec une fenêtre de $\tau = 1h$ et un intervalle de temps $\lambda = 30$mins, utilisant un PC standard. La figure A.3 indique le nombre d'arêtes vues dans une fenêtre, environ $m = 20.10^3$ par flux, sur 48 points. Pour les fenêtres indépendantes à 24, nous lisons environ 48.10^4 arêtes, et globalement environ 2.10^6 arêtes. Les Réservoirs sont de taille $k = 400$ et nous sauvegardons en moyenne 100 nœuds et arêtes, soit $4.48.100 \simeq 2.10^4$ d'arêtes, soit une compression de 100. Pour $\gamma = 0.8$, la taille minimale d'un cluster est de $m/\gamma.k \simeq 60$. Notez que k est proche de $\sqrt{m}$.

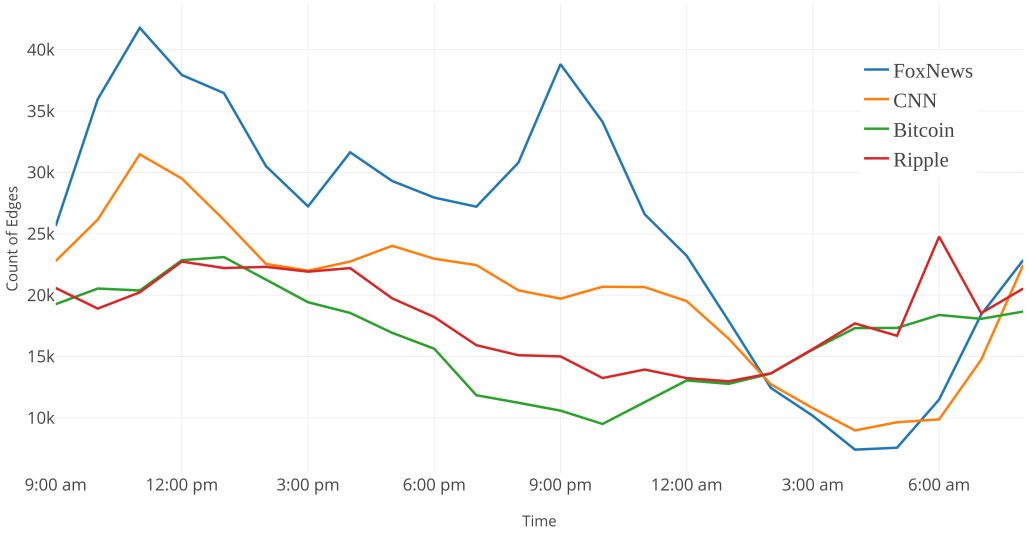

FIGURE 6.2: Nombre d'arêtes par fenêtre d' $1h$, dans 4 flux de données pendant $24h$

Dans [62], l'idée principale de ces travaux repose sur le fait que la stratégie d'échantillonnage est dépendante du domaine. Nous avons une approche différente, nous décidons de garder la même stratégie d'échantillonnage indépendamment du domaine, cependant nous mettons l'accent sur la méthode d'intégrations des flux.

1. En utilisant un programme disponible sur https ://github.com/twitterUP2/stream qui prend certains tags, un réservoir de taille k, une fenêtre de taille τ, un interval λ et on sauvegarde les grandes composantes connectées des Réservoirs R_t.

6.2 Corrélation de nœuds entre différents flux

La corrélation classique, aussi appelée corrélation de Pearson $p(X, Y)$ de deux variables aléatoires X, Y de la moyenne μ et de l'écart-type σ est :

$$\frac{I\!E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X . \sigma_Y} \tag{6.1}$$

Comment peut-on l'étendre aux graphes ? On pourrait choisir quelques statistiques sur les graphes et prendre les corrélations entre les paramètres statistiques. Les graphes sociaux ont cependant des statistiques très similaires et pourtant le noyau de l'information semble se cacher dans la structure de leurs clusters.

Compte tenu de deux graphes G_1 et G_2, une première approche de leur corrélation de contenu serait de considérer la similarité Jaccard [2] $J(V_1, V_2)$ sur les domaines des deux graphes. Cette définition à plusieurs inconvénients : elle est indépendante des structures des graphes, elle est très sensible au bruit, et n'est pas bien adaptée lorsque les tailles sont très différentes. On doit également stocker l'ensemble des arêtes de l'ensemble des graphes.

Nous proposons plutôt l'approche suivante : nous estimons d'abord les composantes denses (clusters ou communautés) en utilisant l'échantillonnage uniforme sur les arêtes des fenêtres coulissantes et appliquons la similarité Jaccard uniquement à ces grandes composantes denses. Cette approche exploite la structure des graphes, est insensible au bruit et s'adapte bien lorsque les graphes ont des tailles différentes.

Soit $C_i = \bigcup_j C_{i,j}$ l'ensemble des clusters du graphe G_i, pour $i = 1$ ou 2.

Definition 8. *La corrélation de deux graphes* $\rho = J(C_1, C_2)$.

Cette définition correspond à deux graphes statiques.

Pour deux flux dynamiques $G_1(t)$ et $G_2(t)$ qui partagent une échelle de temps, nous généralisons C_i à :

$$C_i(t) = \bigcup_{t' \leq t} \bigcup_j C_{i,j}(t') \tag{6.2}$$

Definition 9. *La corrélation de deux flux de graphes* $G_1(t)$ *et* $G_2(t)$ *est :*

$$\rho(t) = J(C_1(t), C_2(t)) \tag{6.3}$$

2. La similarité Jaccard ou Index entre deux ensembles A et B est $J(A, B) = |A \cap B|/|A \cup B|$. La distance Jaccard est de $1 - J(A, B)$.

Nous pouvons affiner la corrélation et définir une corrélation amortie pour donner plus d'importance aux composantes récentes.

Dans cette section, nous donnons une solution algorithmique pour les graphes présentés sous forme de flux d'arêtes qui s'échelonnent lorsque nous considérons les graphes dynamiques. La figure 6.3 donne les trois principales corrélations obtenues avec $\rho(t)$ sur les 6 possibles, et la figure 6.4 nous donnes la corrélation moyenne obtenue par $\rho'(t)$ et définie par :

$$\rho'(t) = (\rho(t-1) + \rho(t) + \rho(t+1))/3 \tag{6.4}$$

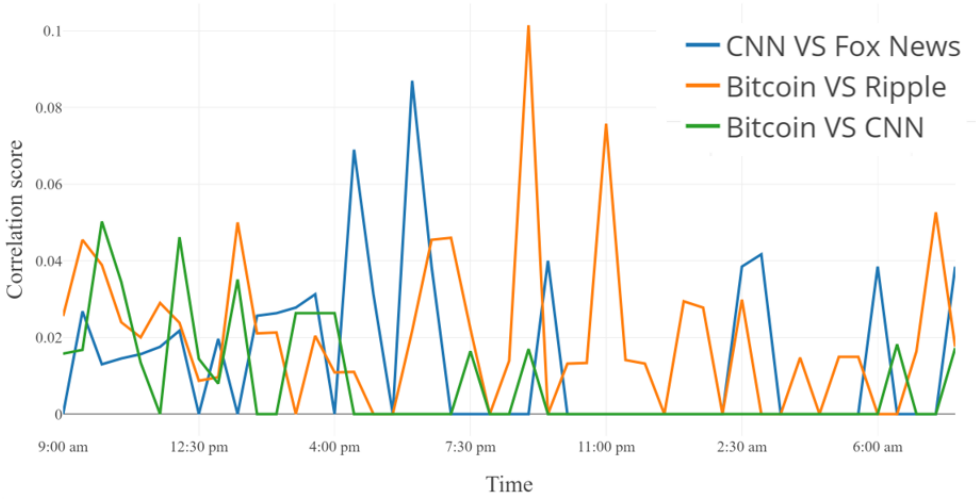

FIGURE 6.3: Corrélation contenu en ligne pendant 24h

Comme nous pouvons nous y attendre la corrélation $\rho(t)$ est très discontinue comme en témoigne la figure 6.3. Cependant en utilisant la version moyenne $\rho'(t)$ la corrélation est plus lisse, comme nous le montre la figure 6.4. La valeur maximale obtenue avec $\rho(t)$ est de 1% et elle est de 0.5% avec la corrélation moyenne. Les corrélations sont toujours petites comme en témoigne la matrice de corrélation et nous avons connu de très petits changements dans les corrélations $\rho(t)$ et $\rho'(t)$. Le spectre des Réservoirs, c'est-à-dire la taille des grandes composantes connectées, est un autre indicateur intéressant. Pour le flux #Bitcoin, il y a une composante unique très grande.

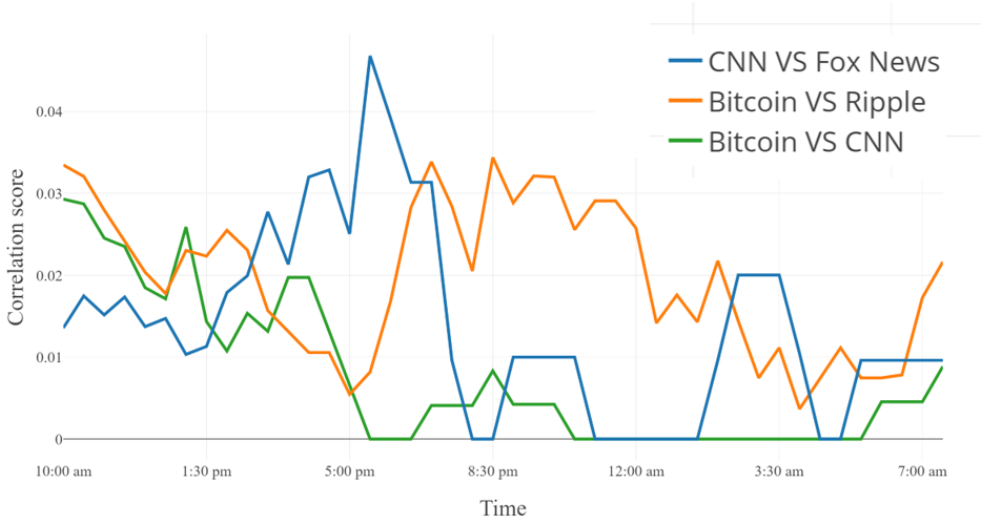

FIGURE 6.4: Corrélation moyenne du contenu en ligne pendant 24h

Considérons deux flux $G_1(t)$ et $G_2(t)$ qui partagent la même horloge. Supposons que $G_1(t)$ soit une stratégie par étapes Δ_1 sur un cluster S_1 et $G_2(t)$ soit une stratégie par étapes Δ_2 sur un cluster S_2. Soit $\rho^* = J(S_1, S_2)$. Comment estimons-nous leur corrélation ?

Soit $C_i(t) = \bigcup_{t' \leq t} \bigcup_j C_{i,j}(t')$ l'ensemble des grands clusters $C_{i,j}(t')$ au moment $t' \leq t$ du graphe G_i, pour $i = 1$ ou 2. Considérez l'algorithme suivant pour calculer $\rho(t)$:

Algorithme en ligne A_3 pour ρ. A l'instant $t + \lambda$, calculez l'augmentation δ_i de la taille de $C_i(t+\lambda)$ pour $i = 1, 2$ de $C_i(t)$, et δ' de la taille de $C_1(t+\lambda) \cap C_2(t+\lambda)$. Supposons $\rho(t) = I/U$ où $I = |C_1(t) \cap C_2(t)|$ et $U = |C_1(t) \cup C_2(t)|$. Alors :

$$\rho(t + \lambda) = \rho(t) + \frac{U.\delta' + I.\delta' - I.(\delta_1 + \delta_2)}{U.(U + \delta_1 + \delta_2 - \delta')} \tag{6.5}$$

Remarquons :

$$\rho(t) + \frac{U.\delta' + I.\delta' - I.(\delta_1 + \delta_2)}{U.(U + \delta_1 + \delta_2 - \delta')} = \frac{I}{U} + \frac{U.\delta' + I.\delta' - I.(\delta_1 + \delta_2)}{U.(U + \delta_1 + \delta_2 - \delta')}$$

$$= \frac{I.(U + \delta_1 + \delta_2 - \delta')) + U.\delta' + I.\delta' - I.(\delta_1 + \delta_2)}{U.(U + \delta_1 + \delta_2 - \delta')}$$

$$= \frac{U.(I + \delta')}{U.(U + \delta_1 + \delta_2 - \delta')} = \frac{(I + \delta')}{(U + \delta_1 + \delta_2 - \delta')} = \rho(t + \lambda)$$

Il suffit de remarquer que $(I + \delta')$ est la taille de la nouvelle intersection des $C_i(t + \lambda)$ pour $i = 1, 2$ et $(U + \delta_1 + \delta_2 - \delta')$ est la taille de l'union des $C_i(t + \lambda)$ pour $i = 1, 2$. Les δ_i, δ' sont calculés par des opérations standard sur les ensembles.

Théorème 4. *Soit $G_1(t)$ et $G_2(t)$ deux flux d'arêtes ayant chacun deux clusters S_1, S_2 tels que $|S_i| \geq m/\gamma.k$ pour $i = 1, 2$ et $J(S_1, S_2) = \rho^*$. L'Algorithme 3 calcule $\rho(t)$ tel que $Prob_{\Omega_t}[|\rho(t) - \rho^*| \leq \epsilon] \geq 1 - \delta$.*

Démonstration : Pour chaque fenêtre, on a d'après les résultats de la section 4.3.1 une bonne approximation de chaque cluster dans chacun des flux (Lemme 2) : V_{S_1} est une bonne approximation de S_1. Après Δ_1/τ essais indépendants, $V_1 = \bigcap_i V_{S_{1,i}}$ sera une meilleure estimation de S_1.

De même pour S_2, pour $S_1 \cup S_2$ et pour $S_1 \cap S_2$. On en déduit que $\rho(t) = J(V_1, V_2)$ sera une (ϵ, δ) approximation de ρ^*.

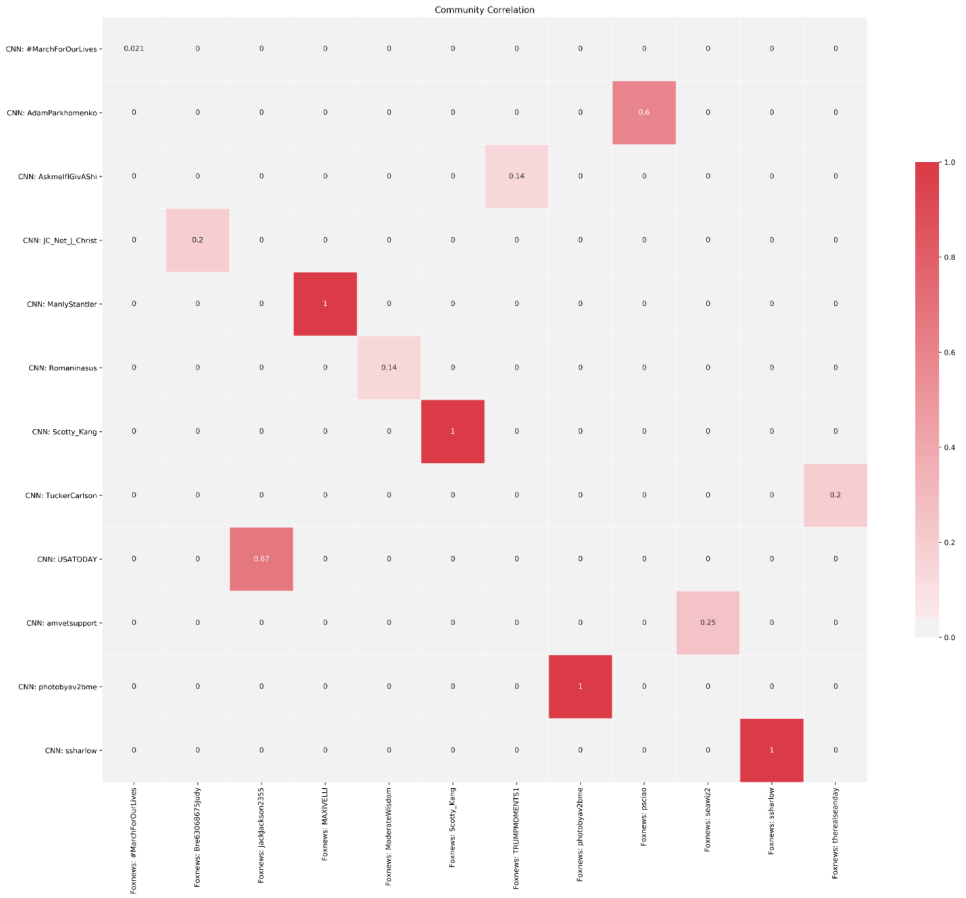

FIGURE 6.5: Composantes dans l'intersection des flux CNN, Foxnews

6.2.1 Phylogénie

Étant donné une matrice de corrélation A_t entre les flux, la méthode Neighbor Joining [56], construit un arbre T avec les valeurs des arêtes, de telle sorte que chaque flux apparaisse comme une feuille dans T et que la distance $d(i, j)$ entre deux flux dans l'arbre soit approximativement la distance définie par la matrice de corrélation, c'est-à-dire $d(i, j) \simeq 1 - A(i, j)$. Cette construction de l'arbre phylogénique suppose une propriété additive des distances, mais il existe toujours une solution approximative.

En quoi la phylogénie est plus intéressante que la matrice ? L'intérêt de la phylogénie, est que nous n'avons pas besoin de suivre les n^2 distances mais seulement quelques distances. La matrice que nous allons suivre sera creuse, mais cela ne nous empêche pas d'avoir des estimations sous forme d'un arbre.

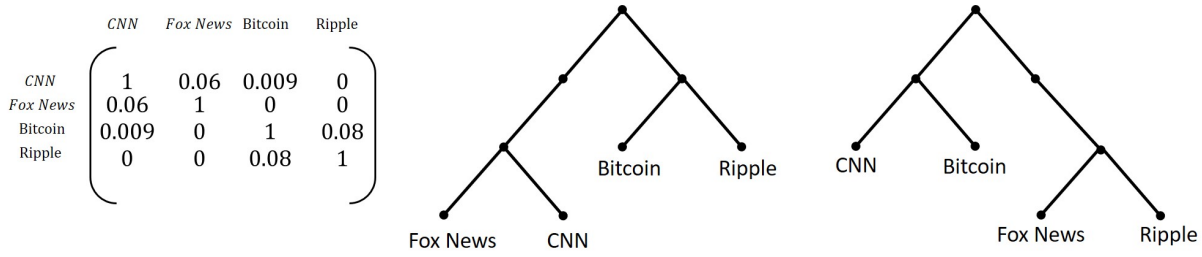

FIGURE 6.6: Arbre phylogénique T (centre) à partir de la matrice de corrélation, et un autre arbre proche T' (droite)

Dans une phase d'apprentissage terminée au temps t, on construit l'arbre T. Plus tard, nous avons une matrice différente A' et un arbre différent T' comme dans la Figure 6.6. Nous pouvons donc facilement détecter de petits ou de grands changements dans l'arbre T. Étant donné un flux, les voisins d'une feuille T sont les flux les plus proches dans T, que nous utilisons dans la recherche par corrélation.

Nous avons une représentation dynamique des distances entre les flux, que nous utilisons dans la section suivante.

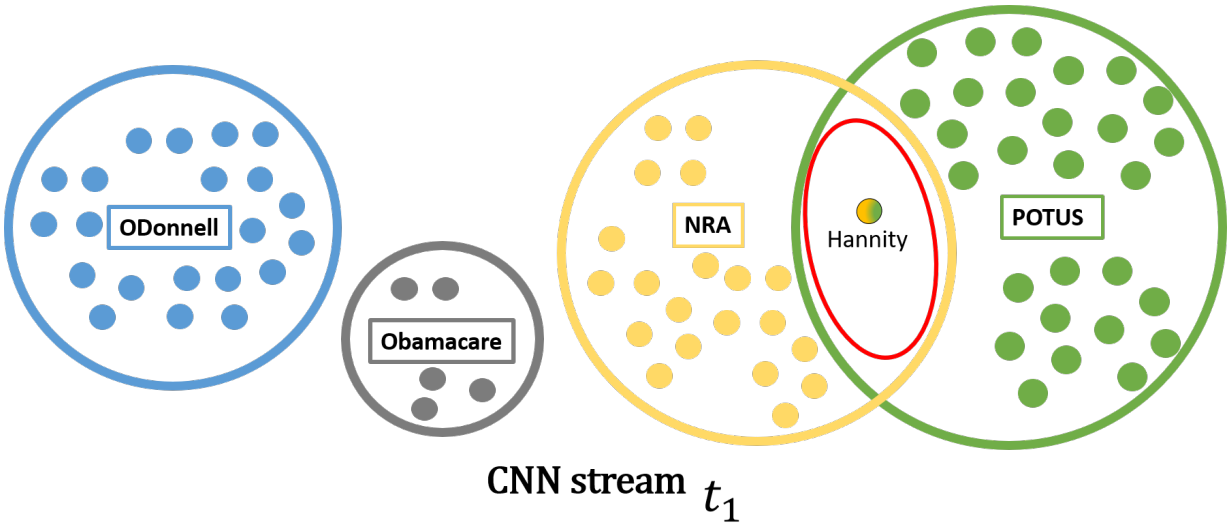

FIGURE 6.7: Noeuds présent dans une intersection de clusters pour l'entrée CNN, POTUS, NRA

6.3 Moteur de recherche basé sur la corrélation des flux de graphes

Nous allons introduire une notion de distance entre les flux, les clusters et les tags, à partir des clusters présents dans les différents flux. Pour cela nous devons avoir une distance entre les flux, c'est pour cette raison que nous allons utiliser la phylogénie. Le but est de créer un moteur de recherche, où étant donné un ensemble de tags, nous allons trouver les tags les plus proches.

Nous avons stocké l'historique des grands clusters pour chaque flux, c'est-à-dire l'ensemble des nœuds des clusters. Pour une requête de recherche définie par un ensemble de tags, la réponse à la requête est l'ensemble des tags les plus corrélés. Nous avons d'abord besoin d'une définition de la corrélation entre un tag et un ensemble de tags. Étant donné un flux, nous devons trouver d'autres flux proches et utiliser la méthode Phylogénie standard.

6.3.1 Corrélation entre tags au temps t

Nous étendons la corrélation aux tags, c'est-à-dire aux éléments des clusters. Étant donné un tag σ, c'est un élément d'un cluster, le nom d'un cluster ou le nom d'un flux. Chacun des clusters porte le nom du noeud de plus haut degré. Les tables *Stream, Clusters, Nodes* permettent de détecter chacun des cas. Par construction, tous les tags ont au moins une composante à laquelle ils appartiennent.

La figure 6.7 indique comment construire un DAG (Directed Acyclic Graph) avec des

coûts, à partir de l'arbre phylogénique introduit à la section précédente, pour définir la distance entres tags. Chaque flux est une feuille de l'arbre phylogénique : on place tous les clusters d'un même flux comme des nœuds connectés au flux par des arêtes de poids 0, puis tous les tags d'un cluster comme des nœuds connectés à chaque cluster par des arêtes de poids 0. Un tag peut appartenir à plusieurs clusters.

La corrélation entre deux clusters est représentée par une intersection entre les nœuds clusters. Un chemin μ entre deux tags σ_1 et σ_2 est un chemin dans le DAG de la figure 6.8. Le chemin peut traverser deux clusters corrélés C_1, C_2, avec un coût $1 - \rho(C_1, C_2)$. Le coût $cost(\mu)$ d'un chemin μ est la somme des coûts des arêtes et des nœuds corrélés : $d(\sigma_1, \sigma_2, \mu)$.

La distance entre deux tags σ_1 et σ_2 est :

$$\mathsf{dist}(\sigma_1, \sigma_2) = Min_\mu \;\; d(\sigma_1, \sigma_2, \mu)$$

Cette définition se généralise à deux nœuds arbitraires du DAG. On peut aussi considérer uniquement les fenêtres récentes et introduire aussi un coût amorti pour privilégier les clusters récents.

Recherche par algorithme de corrélation $A_4(\sigma_1, ...\sigma_l)$:

— Pour chaque tag σ_i, trouvez les tags σ qui minimisent :

$$\sum_{i=1,...l} \mathsf{dist}(\sigma, \sigma_i)$$

— Pour chaque réponse, on peut trouver le chemin μ_i à chacun des tags σ_i, comme explication de la réponse.

Les nœuds de certaines intersections de composantes récentes auront la corrélation la plus élevée et correspondront aux nœuds qui apparaîtront dans les premiers éléments de la réponse. L'explication de la recherche sera portée par ces clusters C_{i,t_i}, qui sont associés aux tags donnés. Dans l'exemple (voir figure 6.8), la première réponse #Hannity appartient à deux clusters CNN, NRA et $POTUS$.

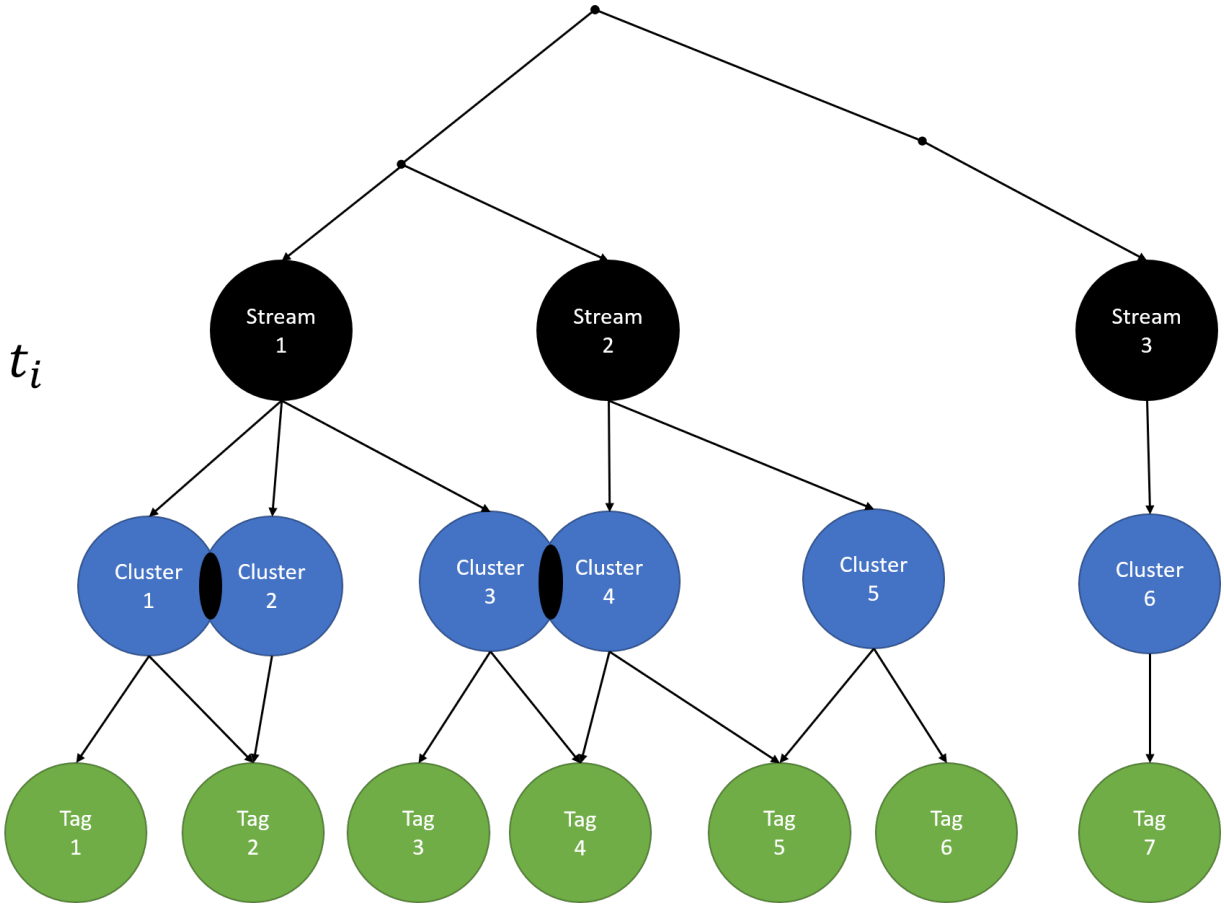

FIGURE 6.8: DAG de recherche généralisé

Nous avons 4 cas possibles qui sont illustrés dans la figure 6.8 :

— Cas 1 : σ_2, σ_3 sont dans le même flux mais dans des clusters différents

— Cas 2 : σ_3, σ_5 sont dans deux flux différents mais sont dans des clusters qui s'intersectionnent

— Cas 3 : σ_1, σ_2 sont dans le même flux et dans des clusters qui s'intersectionnent

— Cas 4 : σ_1, σ_7 sont dans des flux différents et dans des clusters différents

6.3.2 Résultats expérimentaux d'un moteur de recherche

Dans le premier exemple, on nous donne le tag σ_1=CNN. Comme l'indique le tableau 6.1, les réponses sont les nœuds de haut degré de la composante la plus récente du flux CNN. Notez que les réponses dépendent de t, pour les deux exemples $t = 12h$ et $t = 24h$. Pour les tags σ_2=CNN, POTUS, on récupère les composantes les plus récentes et dans ce cas, POTUS appartient à une composante de CNN : on sort les noeuds de haut degré de

cette composante.

Pour le tag, σ_3=CNN, POTUS, NRA, nous récupérons les trois composantes les plus récentes et dans ce cas, la composante de POTUS croise la composante de NRA et tous deux sont des composantes de CNN. Nous sortons les nœuds de l'intersection, classés par degré.

Input : CNN		
Ranking	t =12	t = 24
1	#ODonnell	#POTUS
2	#NRA	#NRA
3	#POTUS	#ODonnell
4	#Obamacare	#Tucker
5	#MondayMotivaton	#2A

Input : CNN POTUS		
Ranking	t = 12	t = 24
1	#MondayMotivaton	#Tucker
2	#Hannity	#2A
3	#NRA	#NRA
4	#1A	#1A
5	#Tucker	#Hannity

Input : CNN POTUS NRA	
Ranking	t = 12
1	#Hannity
2	#2A
3	#1A
4	#Clinton
5	#ClintonFoundation

TABLE 6.1: Résultats de la recherche sur les entrées : σ=CNN, σ=CNN, POTUS et σ=CNN, POTUS, NRA

Node	**Description**
#POTUS	Président des États-Unis d'Amérique.
#NRA	National Rifle Association of America.
#ClintonFoundation	La Fondation Clinton est une ONG américaine.
#Clinton	42ème président des États-Unis.
#Hannity	Animateur de talk-show américain, auteur et commentateur politique conservateur.
#ODonnell	Journaliste américain de la presse écrite et de la télévision, co-présentateur de CBS This Morning,
#Tucker	Un commentateur politique conservateur américain pour Fox News.
#Obamacare	Loi sur la protection des patients et les soins abordables, souvent surnommée Obamacare.
#MondayMotivaton	Hashtags pour parler de retour au travail.
#1A	Premier amendement de la Constitution des États-Unis.
#2A	Deuxième amendement de la Constitution des États-Unis.

TABLE 6.2: Description des résultats sur les sorties de la recherche : σ=CNN, σ=CNN, POTUS et σ=CNN, POTUS, NRA

6.4 Contributions dans le domaine

La détection de grands clusters par les algorithmes présentés précédemment nous a permis de définir *l'intégration approchée de plusieurs flux de données*. Nous avons utilisé l'approximation des clusters pour définir la *corrélation de contenu entre des flux* et le principe de la *recherche par corrélation*. La corrélation de contenu définit une distance entre deux flux, et cette distance peut être étendue au domaine des tags avec une technique d'arbre phylogénique. Nous avons introduit la notion de recherche par corrélation basée sur ces distances.

Conclusion

L'objectif de cette thèse est de proposer des algorithmes de détection de clusters dans les graphes sociaux dynamiques. Nous nous sommes attachés à comprendre comment détecter des clusters dans un graphe défini par un flux d'arêtes, sans stocker l'ensemble du graphe. Nous avons étendu l'approche aux graphes dynamiques qui sont définis par les arêtes les plus récentes du flux, ainsi qu'à plusieurs flux.

Dans le modèle dynamique d'un graphe, nous avons montré que l'on détecte un grand cluster avec grande probabilité. Par contre, dans le cas d'un graphe qui suit une distribution de degrés de loi de puissance et une dynamique uniforme, on montre avec grande probabilité que nous n'avons pas de composante géante.

Nous avons également introduit la corrélation de contenu ρ entre deux graphes et son extension $\rho(t)$ entre deux graphes dynamiques, basée sur la similarité de Jaccard de leurs clusters. Nous avons proposé une méthode simple et efficace pour approcher cette corrélation en ligne et montré que, pour les graphes dynamiques qui suivent une loi de puissance, nous pouvons garantir une bonne approximation. Nous présentons un algorithme en ligne (Algorithme 3) qui estime la corrélation. Comme application, nous suivons les flux Twitter et calculons leurs corrélations de contenu en ligne.

Comme nous lisons différents flux d'arêtes dans le modèle de fenêtre, nous ne stockons que les grands composantes connectées à certains moments t_1, t_2, ... Dans nos expériences, nous avons suivi 4 flux Twitter pendant 24h, lisant ainsi 2.10^6 arêtes, mais nous conservons environ 2.10^4 nœuds, soit 1% des données.

À partir de la matrice de corrélation, nous avons obtenu l'arbre phylogénique le plus proche. Nous proposons ensuite une recherche par corrélation où les réponses aux ensembles de mots-clés sont entièrement basées sur les petites corrélations des flux. Les réponses sont ordonnées par les corrélations et les explications peuvent être tracées avec les clusters stockés. Les témoins utilisés comme explications des corrélations sont les clusters stockés.

Les résultats obtenus dans cette thèse ouvrent plusieurs pistes de recherches :

— Moteur de recherche basé sur les corrélations
— Corrélation de flux unaire et binaire [3]
— Lien avec la compression et la reconstruction de graphe

3. Le flux unaire est défini comme un flux de valeur numérique (par exemple le cours du Bitcoin) et le flux binaire est vu comme un flux Twitter (par exemple le flux Bitcoin). Nous cherchons à savoir comment pouvons-nous corréler ces deux flux ? En passant par l'analyse de la composante géante du flux Twitter nous pouvons corréler sa taille avec le cours du Bitcoin.

Bibliographie

[1] Charu C. Aggarwal. On biased reservoir sampling in the presence of stream evolution. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 607–618. VLDB Endowment, 2006.

[2] Charu C. Aggarwal. An introduction to cluster analysis. In *Data Clustering : Algorithms and Applications*, pages 1–28. CRC Press, 2013.

[3] Charu C. Aggarwal and Chandan K. Reddy, editors. *Data Clustering : Algorithms and Applications*. CRC Press, 2014.

[4] Charu C. Aggarwal, Yuchen Zhao, and Philip S. Yu. On clustering graph streams. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*, pages 478–489, 2010.

[5] Nesreen K. Ahmed, Jennifer Neville, and Ramana Kompella. Network sampling : From static to streaming graphs. *ACM Trans. Knowl. Discov. Data*, 8(2) :7 :1–7 :56, June 2013.

[6] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74 :47–97, Jan 2002.

[7] Hélio Almeida, Dorgival Guedes, Wagner Meira, and Mohammed J. Zaki. Is there a best quality metric for graph clusters ? pages 44–59, 2011.

[8] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proc. VLDB Endow.*, 5(5) :454–465, January 2012.

[9] Albert-László Barabási. Scale-free networks : A decade and beyond. *Science*, 325(5939) :412–413, 2009.

[10] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439) :509–512, 1999.

[11] P. Berkhin. A survey of clustering data mining techniques. *Grouping Multidimensional Data*, pages 25–71, 2006.

[12] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. pages 173–182, 2015.

[13] N. Biggs. *Discrete Mathematics*. Discrete Mathematics. OUP Oxford, 2002.

[14] Stefano Boccaletti, Vito Latora, Yamir Moreno, M.and Chavez, and Dong-Uk Hwang. Complex networks : Structure and dynamics. *Physics Reports*, 424 :175–308, 02 2006.

[15] Bela. Bollobas. *Graph theory : an introductory course / Bela Bollobas.* Springer Verlag,, New York, 1979. Includes index.

[16] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25 :163–177, 2001.

[17] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. In *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA*, pages 147–156, 2009.

[18] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization*, APPROX '00, pages 84–95, Berlin, Heidelberg, 2000. Springer-Verlag.

[19] Ashish Chiplunkar, Michael Kapralov, Sanjeev Khanna, Aida Mousavifar, and Yuval Peres. Testing graph clusterability : Algorithms and lower bounds. pages 497–508, 2018.

[20] Fan Chung, Paul Horn, and Linyuan Lu. Percolation in general graphs. *Internet Math.*, 6(3) :331–347, 2009.

[21] Nick Crossley, Christina Prell, and John Scott. Social network analysis : Introduction to special edition. *Methodological Innovations Online*, 4(1) :1–5, 2009.

[22] Easley David and Kleinberg Jon. *Networks, Crowds, and Markets : Reasoning About a Highly Connected World*. Cambridge University Press, New York, NY, USA, 2010.

[23] Michel de Rougemont and Guillaume Vimont. The value of analytical queries on social networks. In *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015*, pages 2374–2383, 2015.

[24] Michel de Rougemont and Guillaume Vimont. Approximate integration of streaming data. In *Actes des 13èmes journées francophones sur les Entrepôts de Données et l'Analyse en Ligne, Business Intelligence & Big Data, EDA 2017, Lyon, France, 3-5 mai 2017*, pages 37–52, 2017.

[25] Michel de Rougemont and Guillaume Vimont. The content correlation of multiple streaming edges. In *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, pages 1101–1106, 2018.

[26] Michel de Rougemont and Guillaume Vimont. Information search, integration, and personalization - 12th international workshop, ISIP 2018, fukuoka, japan, may 14-15, 2018, revised selected papers. 1040, 2019.

[27] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. Efficient densest subgraph computation in evolving graphs. In *Proceedings of the 24th International Conference*

on World Wide Web, WWW '15, pages 300–310, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee.

[28] P Erdös and A Rényi. On random graphs i. *Publicationes Mathematicae Debrecen*, 6 :290–297, 1959.

[29] Paul Erdős and Alfréd Rényi. On random graphs i. *Publicationes Mathematicae (Debrecen)*, 6 :290–297, 1959 1959.

[30] P. Erdős and A Rényi. On the evolution of random graphs. In *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, pages 17–61, 1960.

[31] T. Erdős, P. ; Gallai. Gráfok előírt fokszámú pontokkal. *Matematikai Lapok*, 11 :264–274, 1960.

[32] Hossein Esfandiari, MohammadTaghi Hajiaghayi, and David P. Woodruff. Brief announcement : Applications of uniform sampling : Densest subgraph and beyond. pages 397–399, 2016.

[33] Uriel Feige and Robert Krauthgamer. Finding and certifying a large hidden clique in a semirandom graph. *Random Struct. Algorithms*, 16(2) :195–208, March 2000.

[34] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486, 06 2009.

[35] Bailey Fosdick, Daniel Larremore, Joel Nishimura, and Johan Ugander. Configuring random graph models with fixed degree sequences. *SIAM Review*, 60, 08 2016.

[36] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1) :35–41, 1977.

[37] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12) :7821–7826, 2002.

[38] Pili Hu and Wing Cheong Lau. A survey and taxonomy of graph sampling. *CoRR*, abs/1308.5865, 2013.

[39] Yuheng Hu, Kartik Talamadupula, and Subbarao Kambhampati. Dude, srsly ? : The surprisingly formal nature of twitter's language. pages 244–253, 2013.

[40] Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann, and Mathieu Bastian. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. *PLOS ONE*, 9(6) :1–12, 06 2014.

[41] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings : Good, bad and spectral. *J. ACM*, 51(3) :497–515, May 2004.

[42] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2) :291–307, Feb 1970.

[43] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 631–636, New York, NY, USA, 2006. ACM.

[44] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2nd edition, 2014.

[45] Andrew McGregor. Graph stream algorithms : A survey. *SIGMOD Rec.*, 43(1), 2014.

[46] Andrew McGregor, David Tench, Sofya Vorotnikova, and Hoa T. Vu. Densest subgraph in dynamic graph streams. *CoRR*, abs/1506.04417, 2015.

[47] Stanley Milgram. The small-world problem. *Psychology Today*, 1(1), 1967.

[48] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 29–42, New York, NY, USA, 2007. ACM.

[49] Michael Molloy and Bruce Reed. The size of the giant component of a random graph with a given degree sequence. *Comb. Probab. Comput.*, 7(3) :295–305, 1998.

[50] M. Newman. The structure and function of complex networks. *SIAM Review*, 45(2) :167–256, 2003.

[51] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23) :8577–8582, 2006.

[52] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2) :026113, February 2004.

[53] Mark Newman. *Networks : An Introduction*. Oxford University Press, Inc., 2010.

[54] Gergely Palla, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. *Nature*, 446(7136) :664–667, 2007.

[55] Anil Kumar Patidar, Jitendra Agrawal, and Nishchol Mishra. Analysis of different similarity measure functions and their impacts on shared nearest neighbor clustering approach.

[56] N Saitou and M Nei. The neighbor-joining method : a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4) :406–425, 1987.

[57] Satu Elisa Schaeffer. Survey : Graph clustering. *Comput. Sci. Rev.*, 1(1) :27–64, August 2007.

[58] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. *SOCIOMETRY*, 32(4) :425–443, 1969.

[59] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1) :37–57, March 1985.

[60] S. Wasserman and K. Faust. *Social Network Analysis : Methods and Applications*. Structural Analysis in the Social Sciences. Cambridge University Press, 1994.

[61] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684) :440–442, June 1998.

[62] Muhammad Bilal Zafar, Parantapa Bhattacharya, Niloy Ganguly, Krishna P. Gummadi, and Saptarshi Ghosh. Sampling content from online social networks : Comparing random vs. expert sampling of the twitter stream. *ACM Trans. Web*, 9(3) :12 :1–12 :33, June 2015.

[63] Reza Zafarani, Mohammad Ali Abbasi, and Huan Liu. *Social Media Mining : An Introduction*. Cambridge University Press, New York, NY, USA, 2014.

Liste des Algorithmes

Table des figures

Liste des tableaux

A Expérience sur la dynamique des clusters

Nous avons dans cet exemple le flux Twitter du tag $\#ONPC$. Ce tag est associé à une émission de télévision hébdomadaire française qui a une durée de 3 heures.

Nous capturons le flux pendant 4 heures, en commençant 1 heure avant le programme, les arêtes contenant en leur extrémité le tag $\#ONPC$ sont supprimées du flux. Il y a environ 10^4 tweets avec une moyenne de $2,5$ tags par tweet, soit 25.10^3 arêtes potentielles et 15.10^3 arêtes sans le tag $\#ONPC$. Si nous ne supprimons pas ces arêtes, le nœud $\#ONPC$ dominera le graphe. Nous avons implémenté l'algorithme "Step reservoir sampling" avec les paramètres suivants : $k = 400$, $\tau = 3h$, $\lambda = 15$mins.

Sur 4 heures, il y a 16 intervalles pour $\lambda = 15$mins, et 4 composantes en moyenne, la taille d'une composante est de 8 en moyenne. C'est pourquoi nous stockons environ $16 * 4 * 8 = 512$ éléments, la représentation de la dynamique des communautés. La figure A.1 montre l'évolution des tailles des composantes connectées. Chaque flux peut être stocké sous forme compressée à des moments différents.

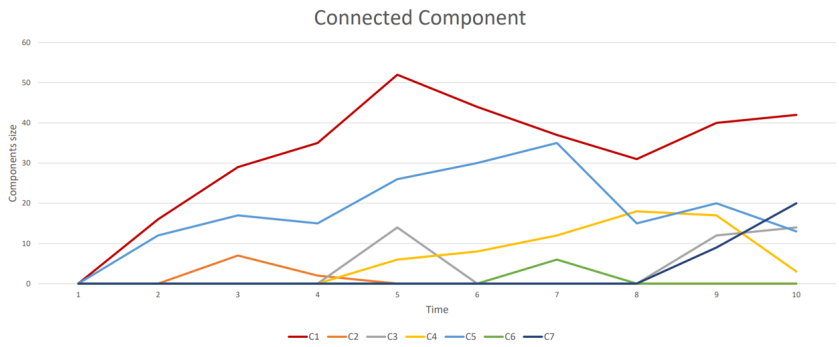

FIGURE A.1: Evolution des la tailles des composantes connectées

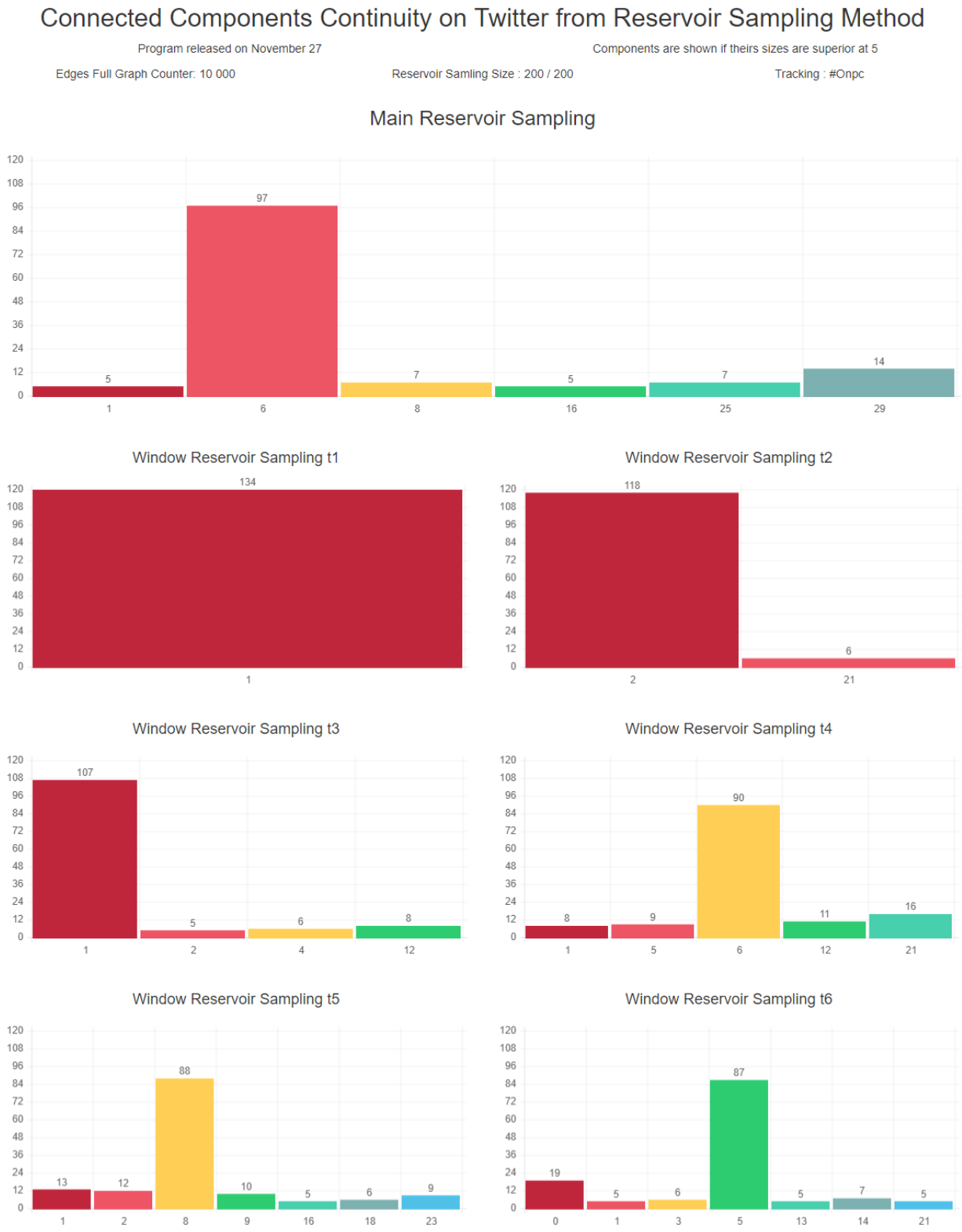

FIGURE A.2: Nombre d'arêtes dans une fenêtre d'1h, dans 4 flux d'arêtes pendant 24h

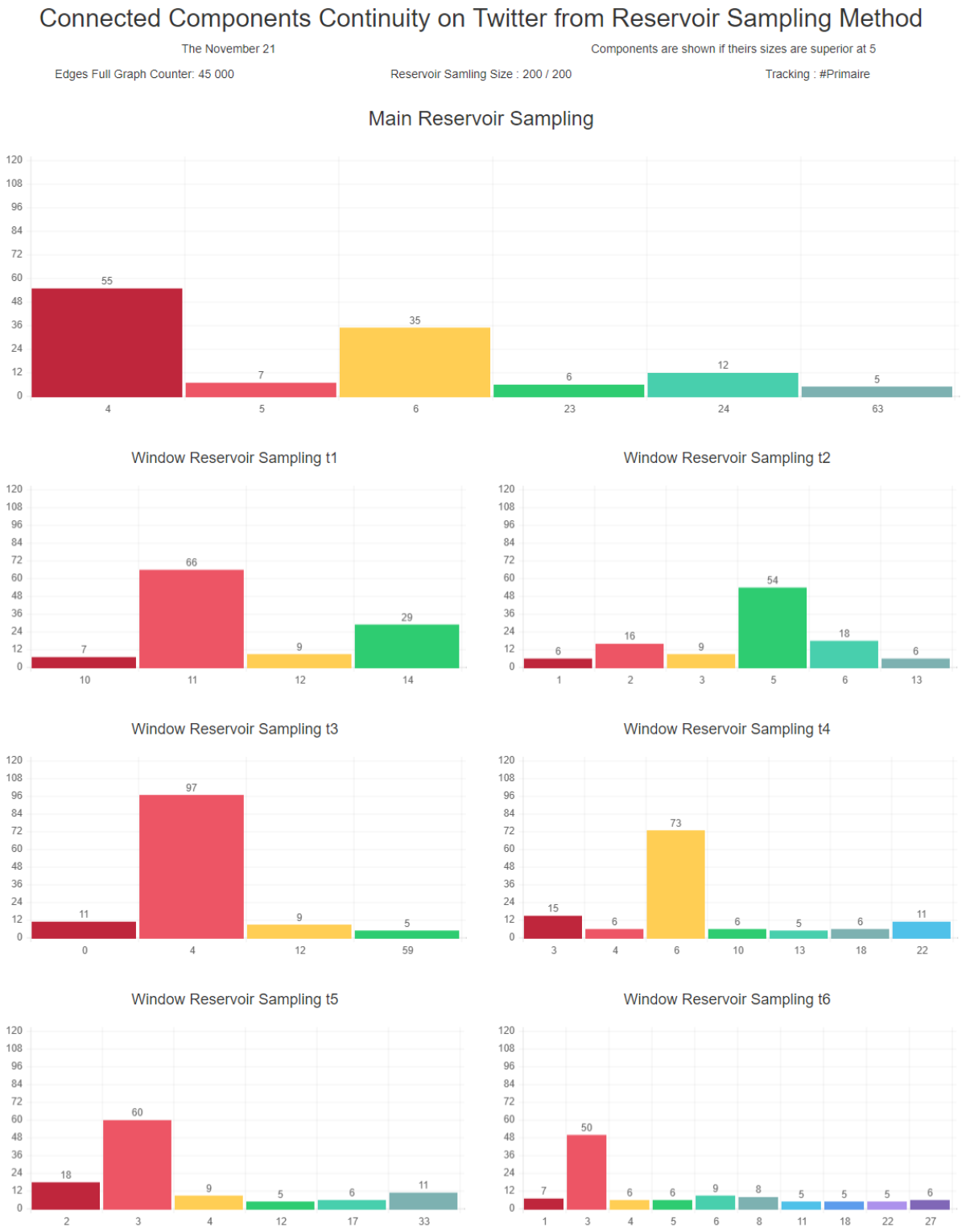

FIGURE A.3: Nombre d'arêtes dans une fenêtre d'1h, dans 4 flux d'arêtes pendant 24h

B Exemple de composantes géantes sur Twitter

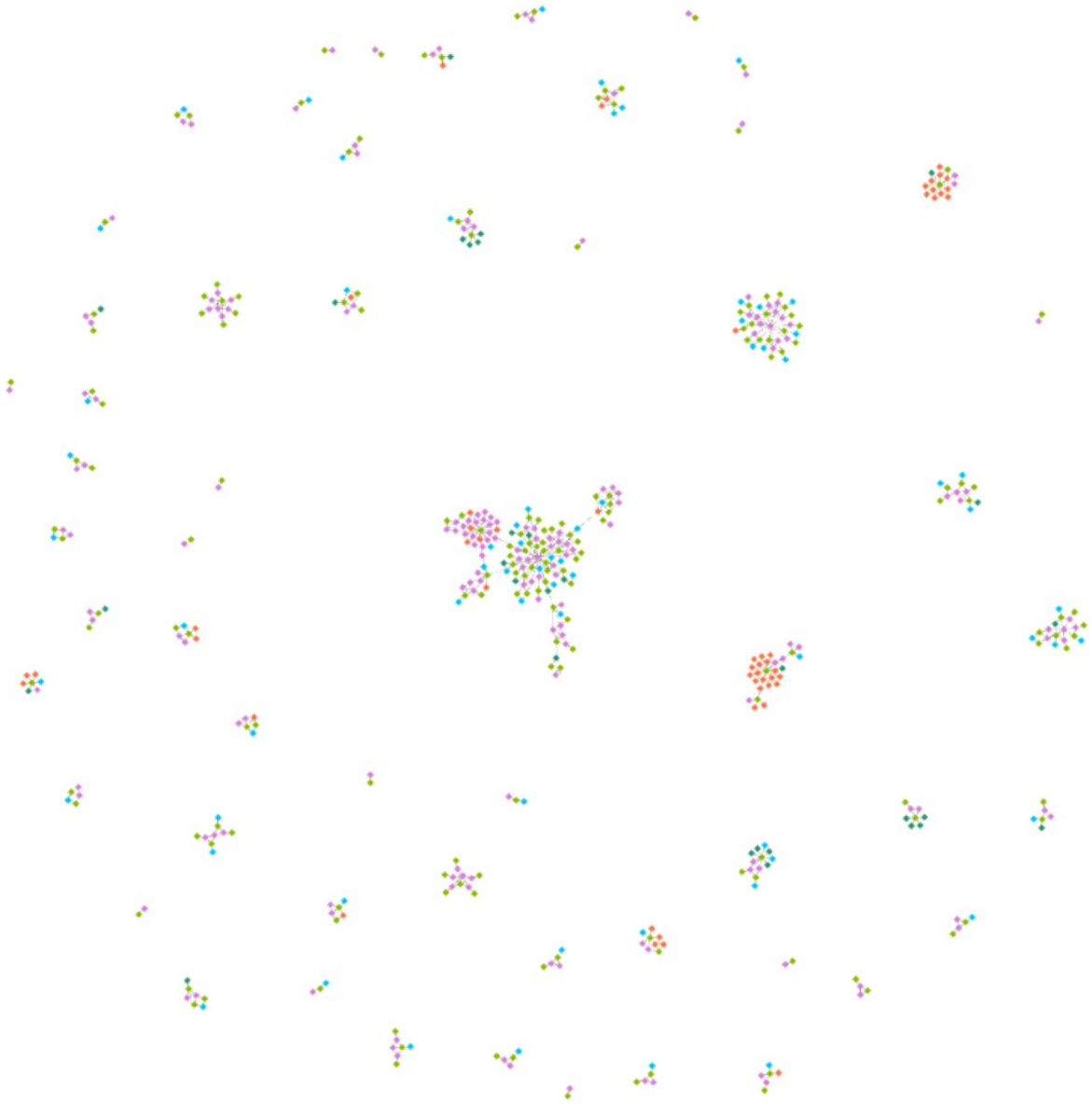

FIGURE B.1: Composante géante issue du flux CNN

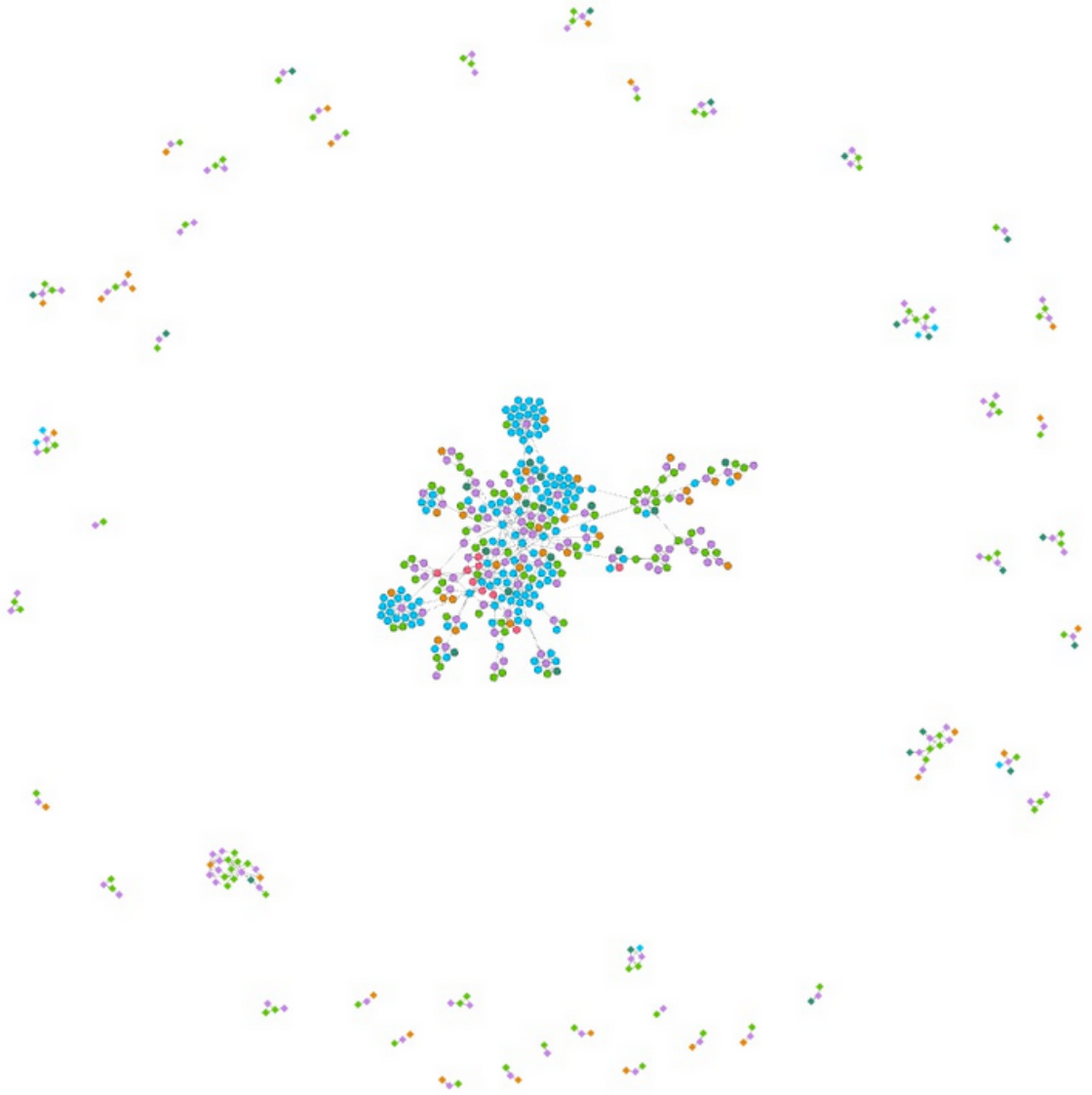

FIGURE B.2: Composante géante issue du flux Bitcoin

C Architecture du Step Continuous Sampling en Python

Nous présentons ci-dessous l'architecture du programme Python que nous avons mis en place. Il est important de noter que lors de nos expériences, nous avons fait une implémentation asynchrone. Cette implémentation nous permet d'être toujours disponible pour écouter le flux Twitter et ainsi ne pas être freiné par les calculs que nous réalisons pour maintenir le réservoir.

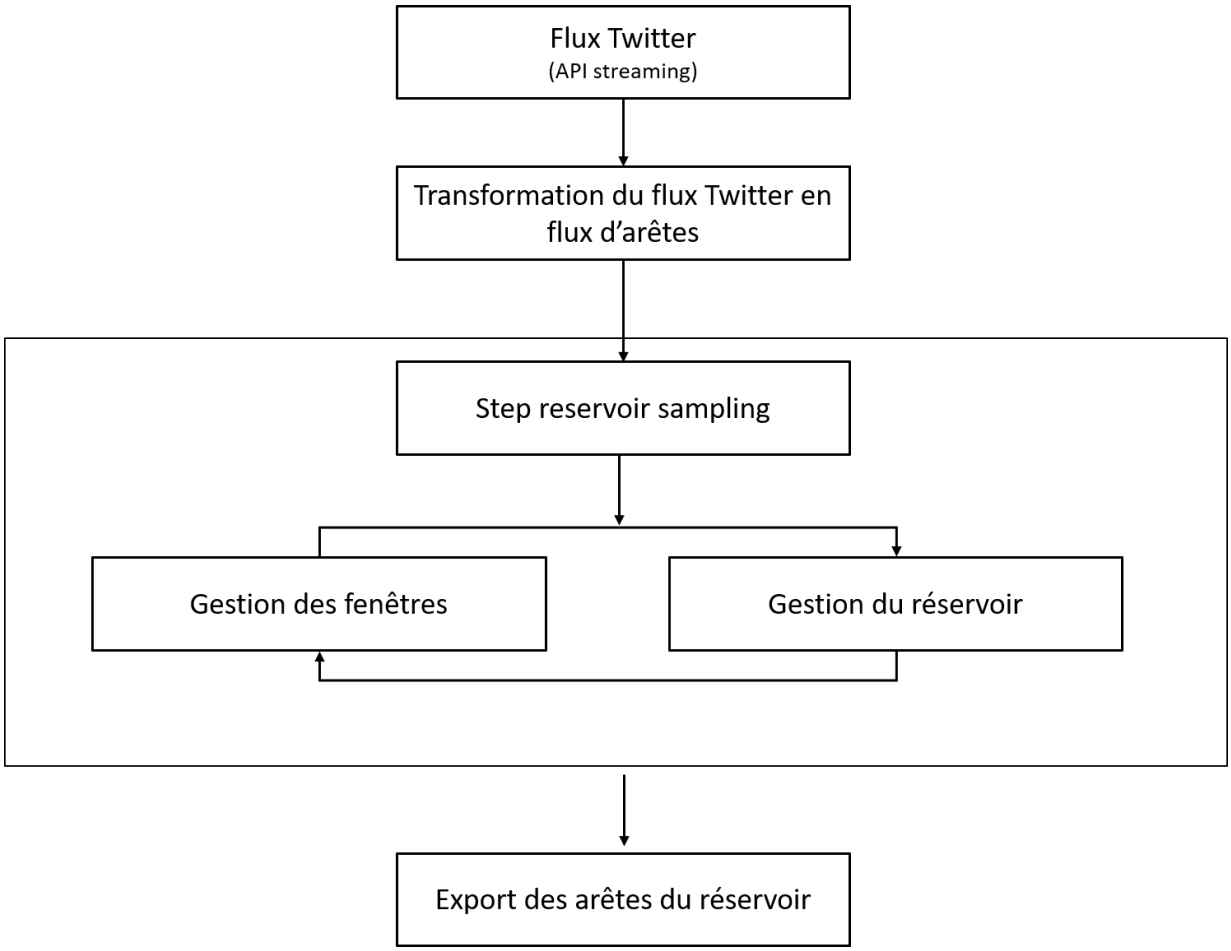

FIGURE C.1: Architecture du programme Python

D Pseudo-code du Step Continuous Sampling

Algorithme 2 Step reservoir sampling

WINDOW_FUNCTION
Require: τ, λ, e_j
$t_i = \tau + \lambda.(i-1)$
$w_i = [t_i - \tau, t_i]$
$M = [0] * \frac{\tau}{\lambda}$
for each e_j arriving from the input stream **do**
if $e_j \in w_i$ **then**
M[-1]+=1
RESERVOIR_FUNCTION(e_j, m_i)
else
$w_i += \lambda$
del m[0]
m.append(1)
for each e_j in the reservoir **do**
if $e_j \notin w_i$ **then**
remove edge into the reservoir
$k_d -= 1$
end if
end for
RESERVOIR_FUNCTION(e_j, m_i, k_d)
end if
end for

RESERVOIR_FUNCTION
Require: k_d, m_i, e_j, k
Reservoir = []
for each e_j arriving from WINDOW_FUNCTION **do**
if $m_i < k$ **then**
add edge at the beginning of the *Reservoir*
$k_d += 1$
else
decide with the probability $\frac{k}{m_i}$ whether to accept the edge e_j
if the edge e_j is accepted **then**
if $k_d > k$ **then**
remove selected edge in the *Reservoir*
and add the accepted edge at the beginning of the *Reservoir*
else
add the accepted edge at the beginning of the *Reservoir*
$k_d += 1$
end if
end if
end if
end for

Résumé :

Nous étudions comment détecter des clusters dans un graphe défini par un flux d'arêtes, sans stocker l'ensemble du graphe. Nous montrons comment détecter de gros clusters de l'ordre de $\sqrt{n}$ dans des graphes qui ont $m = O(n \log(n))$ arêtes, tout en stockant $\sqrt{n \log(n)}$ arêtes. Nous étendons notre approche aux graphes dynamiques définis par les arêtes les plus récentes du flux et à plusieurs flux. Nous proposons des méthodes simples et robustes afin de détecter ces clusters de manière approchée.

Nous définissons la corrélation de contenu de deux flux $\rho(t)$ par la similarité de Jaccard de leurs clusters, dans les fenêtres au temps t. Nous proposons une méthode simple et efficace pour approcher cette corrélation en ligne et montrons que pour les graphes aléatoires dynamiques qui suivent une loi de puissance, nous pouvons garantir une bonne approximation.

Une des applications est l'analyse des flux Twitter. Nous calculons les corrélations de contenu de ces flux en ligne. Nous proposons ensuite une recherche par corrélation où les réponses aux ensembles de mots-clés sont entièrement basées sur les petites corrélations des flux. Les réponses sont ordonnées par les corrélations, et les explications peuvent être tracées avec les clusters stockés.

Descripteurs : Algorithmes de streaming, graphes dynamiques, Clustering, Approximation

Abstract :

We study how to detect clusters in a graph defined by a stream of edges, without storing the entire graph. We show how to detect large clusters in the order of $\sqrt{n}$ in graphs that have $m = O(n \log(n))$ edges, while storing $\sqrt{n \log(n)}$ edges. We extend our approach to dynamic graphs defined by the most recent stream of edges and multiple streams. We propose simple and robust methods based on the approximation to detect these clusters.

We define the content correlation of two streams $\rho(t)$ is the Jaccard similarity of their clusters in the windows before time t. We propose a simple and efficient method to approach this online correlation and show that for dynamic random graphs that follow a power law, we can guarantee a good approximation.

As an applications we follow Twitter streams and compute their content correlations online. We then propose a search by correlation where answers to sets of keywords are entirely based on the small correlations of the streams. Answers are ordered by the correlations, and explanations can be traced with the stored clusters.

Keywords : Streaming algorithms, Dynamic graphs, Clustering, Approximation

Nota : cette page, dernière de couverture, sera retournée avant reliure.