Thèse de doctorat / 2020

Université Panthéon - Assas

École doctorale d'économie, gestion, information et communication

Thèse de doctorat en informatique

soutenue le

Composantes géantes sur des flux de données

UNIVERSITÉ PARIS II
PANTHÉON-ASSAS

Achraf Lassoued

Sous la direction de Michel De Rougemont

Rapporteur :

Mme. Vilnat Anne, Professeur associé à l'Université Paris-Saclay
M. Laurent Dominique, Professeur émérite à l'Université Cergy-Pontoise

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

Ce manuscrit constitue la synthèse de trois ans de doctorat effectué au Centre de Recherche en Économie et Droit (*CRED*) à l'Université Paris II Panthéon-Assas.

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.

J'adresse un remerciement particulier à Madame Isabelle Bril, Directrice de recherches (LACITO-CNRS) et Directrice d'Etudes à l'EPHE (Typologie et Typologie des langues austronésiennes), pour son rôle déterminant dans l'encadrement et le financement d'une partie de ma thèse.

Je remercie sincèrement Madame Anne Vilnat , Professeur à l'Université Paris-Saclay, ainsi que Monsieur Laurent Dominique, Professeur émérite à l'Université de Cergy Pantoise, pour l'intérêt qu'ils ont témoigné à l'égard de mon travail en acceptant la charge de rapporteurs.

J'exprime ma reconnaissance à Madame Céline Chevalier, Maître de conférences à l'Université Paris 2, et Monsieur Nicolas Spyratos, Professeur émérite à l'Université Paris-sud pour leurs participation au jury en tant qu'examinateurs.

Je tiens également à remercier mes collègues, camarades du lycée Saint Nicolas et l'EPF, pour tout le soutien 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 ma soeur qui a été solidaire, encourageante et patiente, au cours de cette thèse.

Je vous remercie.

Achraf Lassoued
Université Panthéon-Assas

Résumé :

Nous étudions des données de nature diverse sous forme de flux, en particulier :

- Base de données,
- Réseaux sociaux,
- Données de texte.

Pour une base de données qui suit un schéma relationnel, un schéma d'analyse *OLAP* (Online Analytical Processing) définit une des tables de la base de données comme une table d'analyse. Nous supposons que les tuples de la table d'analyse arrivent sous forme d'un flux. Nous étudions l'approximation des requêtes *OLAP*, en échantillonnant de manière non uniforme les tuples du flux sans stocker les données d'analyse et donnons un modèle de préférence dans ce cadre.

Dans le cas du réseau social *Twitter*, nous observons un flux de tweets qui contiennent un tag donné et le transformons en un flux d'arêtes d'un graphe. Nous souhaitons étudier l'existence des grands clusters dans le graphe ainsi obtenu. Nous proposons une méthode d'échantillonnage uniforme qui va associer au graphe un sous-graphe aléatoire et étudions les composantes géantes de ce sous-graphe aléatoire comme témoin des grands clusters du graphe d'origine.

Pour un flux de texte, nous considérons les paires de mots dans une phrase lemmatisée comme des arêtes d'un graphe où les nœuds sont les mots. Nous transformons le flux de texte en flux d'arêtes. Nous échantillonnons les arêtes proportionnellement à la similarité *Word2vec* des mots. Nous analysons ensuite les composantes géantes.

Nous étendons les vecteurs *Word2vec* en prenant en compte la morphologie d'une langue, en particulier la structure des préfixes et des suffixes d'un mot. Les nouveaux vecteurs d'un mot ont 3 composantes : un vecteur pour le préfixe, un vecteur pour la racine et un vecteur pour le suffixe. Nous définissons un vecteur probabiliste pour les phrases.

Sur les trois types de données, nous avons échantillonné selon des distributions précises. Les résultats principaux de cette thèse montrent comment approcher les propriétés de ces données avec ces échantillons.

Descripteurs : Algorithmes de streaming, Approximation, Requêtes OLAP, Préférences, Clustering, Graphes dynamiques, TAL, Morphologie

Abstract :

We study different types of data as a stream :

- Databases,
- Social Networks,
- Text data.

For a database that follows a relational schema, an *OLAP* (Online Analytical Processing) analysis schema defines one of the database tables as an analysis table. We suppose that the tuples of the analysis table arrive as a stream. We study the approximation of *OLAP* queries, by sampling with weights the tuples of the stream without storing the analysis data. We give a *preference* model in this context.

For the social network *Twitter*, we observe a stream of tweets that contain any given tag and transform it into a stream of edges of a graph. We study the existence of large clusters in the generated graph. We propose a uniform sampling method that will associate a random subgraph of the graph and study the giant components of this random subgraph as a witness of the large clusters of the original graph.

For a stream of text, we consider the pairs of words in a lemmatized sentence as edges of a graph where the nodes are the words. We transform the stream of text into a stream of edges. We sample the edges proportionally to the *Word2vec* similarity of the words. We then analyze the giant components.

We extend the classical *Word2vec* vectors, in order to take into account the morphology of the words, i.e. the prefixes, roots and suffixes. A new vector has 3 components, one for the prefix, one for the root and one for the suffix. We define a probabilistic vector associated with the sentences.

With the three types of data, we sampled according to specific distributions. The main results of this thesis show how the properties of these data can be approximated with these samples.

Keywords : Streaming algorithms, Approximation, OLAP queries, Preferences, Clustering, Dynamic graphs, NLP, Morphology

Sommaire

1 Introduction

Des flux de données sont disponibles partout autour de nous, soit sous forme de données que nous recherchons (pages web, documents), soit sous forme de données envoyées par une plateforme numérique suite à un abonnement. Par exemple une vidéo envoyée par *Netflix* ou un flux envoyé par un réseau social comme *Twitter*.

Les réseaux sociaux font aujourd'hui partie de notre vie quotidienne après l'apparition de Facebook et Twitter. Les données issues de ces réseaux sont très différentes des données classiques que nous utilisons en exploration de données. Ces réseaux génèrent des flux de données volumineux qui permettent de développer des nouveaux services et de créer des nouvelles valeurs économiques. Dans ce cadre la règle des quatre "V" symbolise ce nouveau monde :

- *Volume :* la quantité volumineuse d'informations.
- *Variété :* l'hétérogénéité des formats, de types, et de qualité des informations.
- *Vélocité :* l'aspect dynamique et temporel des données.
- *Véracité :* le fait que les données contiennent beaucoup de bruit.

Dans cette thèse, on introduit des méthodes qui gèrent des flux volumineux, pour une variété de données, qui évoluent très vite dans le temps et qui ont beaucoup de bruit. Ces méthodes sont basées sur un échantillonnage des données selon différentes lois. La loi la plus simple est la loi uniforme. On analyse les types de données suivants :

- Entrepôts de données au sein d'une base de données.
- Graphes issus des réseaux sociaux .
- Texte.

Notre méthode d'analyse de données s'applique à ces trois types de données et suit les phases suivantes :

- Nous échantillonnons des flux de données avec des distributions particulières pour ne retenir qu'un nombre constant d'échantillons. Pour chaque élément du flux, nous décidons avec une certaine probabilité si l'élément est conservé ou pas. L'ensemble des échantillons est stocké dans un *Réservoir* qui est une structure aléatoire.
- L'analyse de données consiste à observer les composantes géantes du Réservoir.

- On introduit une distance entre ces composantes géantes qui permettra de classifier et de rechercher les propriétés essentielles des données.

Supposons un flux de tuples t associés à un entrepôt de données qu'on cherche à analyser à l'aide de requêtes d'analyse OLAP (Online Analytical Processing) pour certaines *Mesures* M de ces tuples. Nous montrons comment approximer les réponses aux requêtes OLAP en échantillonnant les tuples selon une certaine loi qui dépend de la *Mesure* M. On obtient alors des réponses approximatives en temps constant sans lire tout l'entrepôt de données.

Les réseaux sociaux génèrent des flux de données, plus précisément des flux d'arêtes d'un graphe. À partir de certains mots clés, Twitter génère tous les tweets qui contiennent ces mots clés. Typiquement, nous recevons 10^3 tweets par minute. Une méthode classique transforme le flux de tweets en un flux d'arêtes d'un graphe où les nœuds sont les tags. L'analyse des grandes communautés du graphe Twitter est basée sur la détection de composantes géantes dans un graphe aléatoire. Nous utilisons un échantillonnage de Réservoir et dans ce cas, suivons la loi uniforme. Nous analysons les composantes géantes du Réservoir de taille constante k.

Pour l'analyse de texte, on utilise la technique *Word2vec*, introduite dans [44, 51] qui associe un vecteur de petite dimension à chaque mot. La similarité de deux mots est le produit de deux vecteurs. Un abonnement à un flux RSS d'article d'actualité génère un flux important de texte. Nous utilisons des méthodes classiques pour le prétraitement du texte par exemple la lemmatisation. Nous considérons ensuite les paires de mots d'une même phrase comme des arêtes d'un graphe où les noeuds sont des mots. Le flux de texte est transformé ainsi en un flux d'arêtes. Nous échantillonnons les arêtes avec un poids proportionnel à la similarité *Word2vec* des mots. Nous analysons ensuite les composantes géantes qui deviennent stables dans ce modèle.

Nous étendons l'approche *Word2vec* en utilisant la morphologie d'une langue. Pour un mot qui se décompose selon la structure *préfixe-racine-suffixe*, on peut généraliser la conception de vecteur aux préfixes, aux racines et aux suffixes. Cette analyse a été faite à partir d'une langue naturelle l'*Amis* et nous développons un modèle statistique dans ce cadre.

1.1 Comparaisons avec les autres techniques d'analyse

Il existe de nombreuses méthodes pour analyser les bases de données. En général, les données sont stockées et ne sont pas considérées comme des flux. L'approximation est rarement un sujet central dans le cadre de l'analyse OLAP.

Dans les réseaux sociaux, une méthode classique pour trouver des communautés, c'est-à-dire des sous-graphes denses, est d'utiliser des méthodes spectrales telles que la décomposition par module. La complexité $O(n^3)$ de ces méthodes n'est pas adaptée aux données volumineuses. D'autres méthodes [8, 29, 43, 25] se concentrent sur les solutions algorithmiques pour approximer des sous-graphes denses. Un des buts est d'avoir un algorithme de flux qui utilise une mémoire sous-linéaire pour détecter des sous-graphes denses. Il n'est pas possible de trouver un tel algorithme, pour la notion de complexité dans le pire des cas. Nous présentons cependant, un algorithme efficace pour les graphes qui satisfont une distribution de degré qui suit une loi de puissance.

Pour le texte, nous échantillonnons selon une distribution, analysons des composantes géantes d'un graphe et introduisons une distance entre ces composantes. Nous pouvons ensuite classifier les textes en utilisant l'algorithme classique de *k-means* où k est le nombre de classes. Chacune des k classes est un sujet, avec une composante géante comme représentant. Notre méthode est beaucoup plus simple que les techniques classiques utilisées dans le traitement de texte comme IRaMuTeQ [52] ou LDA (Latent Dirichlet Allocation) [10]. Les mécanismes d'attention [40, 60] suivent les mêmes idées.

1.2 Organisation de la thèse

Dans le chapitre 2 nous introduisons les préliminaires algorithmiques, les concepts de théorie des graphes et en particulier les différents modèles de graphes aléatoires.

Dans le chapitre 3, nous introduisons les préliminaires d'analyse de texte. Nous allons présenter les méthodes classiques utilisées dans le monde du traitement du langage naturel.

Dans le chapitre 4, nous étudions les données d'un entrepôt de données sous forme de flux. Nous montrons comment approximer des requêtes OLAP dans ce cadre en suivant une loi d'échantillonnage particulière, c'est un des résultats principaux de la thèse.

Dans le chapitre 5, nous présentons plusieurs méthodes pour récupérer les données Twitter sous forme d'un flux depuis ce réseau social, puis nous transformons ces données en un flux d'arêtes et échantillonnons ces arêtes avec une loi uniforme pour analyser les composantes géantes du graphe Twitter.

Dans le chapitre 6, nous présentons des méthodes pour analyser du texte sous forme d'un flux. Nous transformons chaque phrase en un flux d'arêtes et échantillonnons ces arêtes selon une distribution *Word2vec*. Un des résultats importants de la thèse est de montrer comment l'échantillonnage selon cette distribution est stable. Nous montrons ensuite comment utiliser les composantes géantes pour classifier le texte.

Dans le chapitre 7, nous montrons que notre méthode peut se généraliser en étudiant la morphologie des mots. Nous présentons un exemple d'analyse sur la langue *Amis*, une langue austronésienne de Taiwan.

Dans le chapitre 8, nous présentons un moteur de recherche basé sur les distances entre les composante géantes introduites dans les chapitres précédents.

Dans le chapitre 9, nous présentons les différentes expériences réalisées dans le cadre de cette thèse.

1.3 Publications

Cette thèse est basée sur les publications suivantes :

- A statistical model for morphology inspired by the Amis language, *Proceedings of the Language, Ontology, Terminology and Knowledge Structures Workshop*, LOTKS 2017 [16] (version étendue de l'article soumise à un journal).
- Topics as giant components in texts generated by a stream, *soumis à une conférence.*
- Preferences as distributions on streaming data, *soumis à International Workshop on Information Search, Integration and Personalization*, ISIP2020.

2 Préliminaires d'algorithmique

Dans ce chapitre nous introduisons des méthodes d'approximation en échantillonnant un flux d'information en particulier pour un flux d'arêtes d'un graphe G. On s'intéresse aux algorithmes qui détectent la présence de cluster dans ce graphe ou qui approximent les clusters. Pour des flux d'arêtes, nous considérons des algorithmes probabilistes qui échantillonnent les arêtes et génèrent des graphes aléatoires.

Nous considérons les composantes géantes du graphe aléatoire et interprétons chaque composante géante comme le représentant d'un cluster. Dans le cas de l'analyse du texte, un algorithme standard transforme le flux de texte en un flux d'arêtes. On réduit l'analyse du texte à l'analyse du graphe aléatoire associé.

Nous distinguons :

- Les méthodes d'échantillonnage,
- Les graphes aléatoires et l'analyse des composantes géantes.

2.1 Échantillonnage des flux de données

Un panorama des différentes techniques d'échantillonnage sur les graphes est présenté dans [34]. Nous considérons seulement l'échantillonnage d'arêtes dans un flux d'arêtes :

- Échantillonnage uniforme par Réservoir,
- Échantillonnage pondéré,
- 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 classique, mais ils nécessitent également de nouvelles innovations significatives.

Notre discussion commence par l'obtention d'un échantillon d'arêtes, lorsque le nombre d'échantillon souhaité est beaucoup plus petit que le nombre total d'éléments dans le flux. Les algorithmes les plus simples consistent à échantillonner le flux de données avec une distribution uniforme.

Pour certaines applications, il peut être souhaitable d'échantillonner selon une distribution biaisée vers des éléments plus récents. Dans les sections suivantes, nous allons discuter

des schémas d'échantillonnage uniforme, pondéré en supposant que chaque élément à un poids et biaisé en favorisant les éléments les plus récents.

2.1.1 Reservoir Sampling

Le Reservoir Sampling est un algorithme probabiliste, introduit par Jeffrey Vitter[62]. 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 garde e_i avec probabilité k/i.
- Si on garde e_i, on enlève une des arêtes du Réservoir (avec probabilité $1/k$) pour faire de la place à e_i.

Algorithme 1 Approche classique du Reservoir Sampling

```
INPUT :
k // size of a Reservoir of size k,
e_1, e_2, ....e_m // Stream
i = 0
for each edge arriving from the input stream do
  i = i+1
  if i ≤ k then
    add the edge to the Reservoir
  else
    decide with the probability k/i 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
```

2.1.1.1 Justification de l'algorithme

La propriété du Réservoir est que chaque arête e_i a une probabilité $\frac{k}{m}$ d'être dans le Réservoir, c'est-à-dire uniforme. On peut le montrer par récurrence sur m. Supposons que la propriété soit vraie à l'étape $m-1$, toutes les arêtes ont la probabilité $\frac{k}{m-1}$. L'élément m a une probabilité $\frac{k}{m}$ de faire partie du Réservoir. Pour les éléments qui en font déjà partie, leur probabilité de rester se décompose en deux événements :

- On ne prend pas l'élément e_m, la probabilité c'est : $1 - \frac{k}{m}$.
- On prend l'élément e_m, avec probabilité $\frac{k}{m}$ et on survit à l'éjection avec $\frac{k-1}{k}$.

La probabilité de rester dans le Réservoir est donc la somme :

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

Pour être dans le Réservoir à l'itération m il faut que deux événements soient réalisés :

- Être dans le Réservoir à l'étape $m-1$ et on sait que par récurrence la probabilité est : $\frac{k}{m-1}$.
- Rester dans le Réservoir : la probabilité de rester est : $\frac{m-1}{m}$ d'après ce qu'on vient de voir.

La probabilité d'être dans le Réservoir à l'étape m est donc :

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

2.1.1.2 Exemple

L'espace probabiliste Ω est déterminé par les choix effectués à chaque étape de l'algorithme. L'algorithme s'applique à tous types de données. On peut présenter l'espace probabiliste sous forme d'un arbre qui reproduit les décisions de l'algorithme.

- Chaque feuille représente un événement, le résultat du tirage.
- Les branches représentent les différentes possibilités quand on rajoute un nouvel élément et la probabilité est indiquée sur chaque branche.

On va présenter le Réservoir quand on a un flux d'arêtes mais on peut aussi l'appliquer à un flux de valeurs. La figure 2.1 montre un exemple où le flux est composé des valeurs $1, 2, 3, 4, 5$ avec $k = 3$. Le premier tirage a lieu quand on lit la valeur 4 et le Réservoir contient $1, 2, 3$, la branche gauche correspond à la situation où on ne garde pas l'élément 4 avec probabilité $1/4$ et les trois autres branches correspondent au cas où on garde la valeur 4 et on enlève 1 sur la branche 2, on enlève 2 sur la branche 3, on enlève 3 sur la branche 4. On répète l'algorithme sur l'entrée 5, et l'arbre global est indiqué sur la figure 2.1.

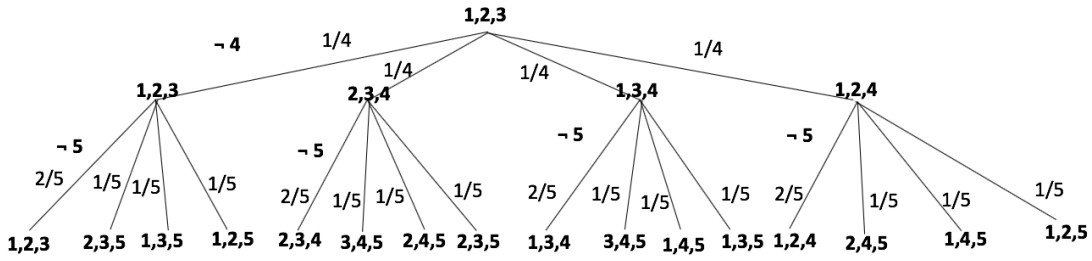

FIGURE 2.1: Espace probabiliste

La propriété essentielle de cette technique, est que tous les sous-ensembles sont équiprobables. Le nombre de sous-ensembles de 3 éléments parmi 1,2,3,4,5 est :

$$\binom{5}{3} = \frac{5!}{3!(5-3)!} = \frac{5*4*3*2}{3*2*2} = 10$$

On va montrer que la probabilité que le Réservoir contient un sous ensemble est égale 1/10 suivant deux explications.

Comment peut-on calculer dans ce tirage la probabilité d'avoir un sous ensemble S ? Pour un sous-ensemble S donné la probabilité qu'un élément donné soit dans le Réservoir est $\frac{k}{m}$. Maintenant si on prend un deuxième élément parmi le deuxième élément de S, la probabilité est $\frac{k-1}{m-1}$ et $\frac{k-2}{m-2}$ pour le troisième. Si on prend $k = 3$ et $m = 5$ la probabilité d'avoir un sous ensemble :

$$Prob(S) = \frac{3}{5} * \frac{2}{4} * \frac{1}{3} = \frac{3*2*1}{5*4*3} = \frac{6}{60} = \frac{1}{10}$$

La probabilité qu'un élément soit dans le Réservoir est 3/5, la probabilité qu'un deuxième élément soit dans le Réservoir est 2/4 et la probabilité qu'un troisième élément soit dans le Réservoir est 1/3. La probabilité d'avoir un sous-ensemble à trois éléments est bien le produit c'est à dire 1/10, ce qui correspond aux probabilités observées dans la figure 2.1.

L'arbre de la figure 2.1 a 16 feuilles et chaque noeud a 4 successeurs. Parmi les 16 feuilles, 4 ont une probabilité de 1/10 et 12 autres ont une probabilité de 1/20. Il existe 4 sous-ensembles parmi les 10 qui ont une seule feuille et une probabilité de 1/10 et 6 autres qui ont 2 feuilles, chacune de probabilité 1/20. Chaque sous-ensemble parmi les 10 a donc bien la même probabilité de 1/10

Maintenant si on regarde le flux $1, 2, 3, 4, 5.....n$ avec $k = 3$, on obtient l'arbre de la figure 2.2 qui montre le cas général.

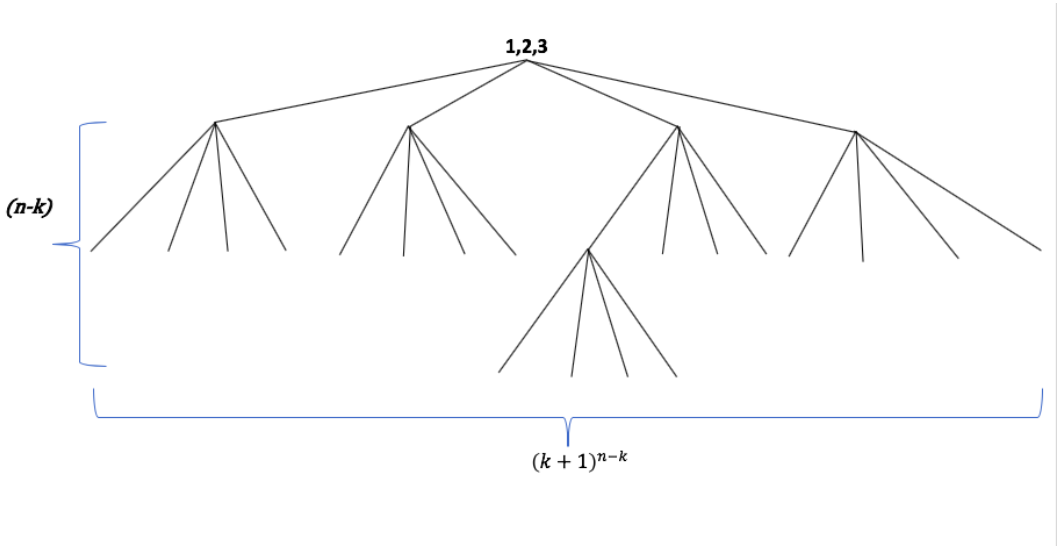

FIGURE 2.2: Espace probabiliste cas général

2.1.2 Réservoir pondéré

Dans le cas des arêtes e avec des poids w_e, l'échantillonnage du Réservoir pondéré conserve k arêtes avec la probabilité $\left(\frac{k*w_e}{\sum w_e}\right)$. Soit $e_1, e_2, ...e_m$ un flux d'arêtes, soit $w_1, w_2, ...w_m$ le poids qui correspond à chaque arête et $R_m = e_{i_1}, e_{i_2}, ...e_{i_k}$ le Réservoir à l'état m de taille k.

Lemme 1. *Pour tout m la probabilité qu'une arête e_i soit dans le Réservoir R_m est*

$$\frac{k * w_i}{\sum\limits_{i=1.....m} w_i}.$$

Comme dans le cas du Reservoir Sampling, on va montrer ce résultat par récurrence. On suppose que c'est vrai pour l'étape m et on veut démontrer que c'est vrai à l'étape $m+1$.

Démonstration. Supposons que

$$Prob[e_i \in R_m] = \frac{k * w_i}{\sum\limits_{i=1.....m} w_i}$$

Algorithme 2 Approche classique du Reservoir pondéré

INPUT :
k // size of a Reservoir of size k,
$e_1, e_2, ...e_m$ // stream of edges,
$w_1, w_2, ...w_m$ // weight of edges
$i = 0$
for each edge arriving from the input stream **do**
  $i = i+1$
  if $i \leq k$ **then**
    add the edge to the Reservoir
  else
    $j = random(0, 1)$
    decide with the probability

$$\frac{K \cdot w_i}{\sum_{j \leq i} w_j}$$

    whether to insert 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

en suivant l'algorithme, la probabilité que e_{m+1} soit dans R_{m+1} est :

$$Prob[e_{m+1} \in R_{m+1}] = \frac{k * w_{m+1}}{\sum_{i=1.....m+1} w_i}$$

La probabilité qu'une arête soit dans le Réservoir à l'étape $m+1$, est la probabilité que cette arête soit à l'étape m et qu'elle survive au tirage de l'arête e_{m+1}. La probabilité de survivre est similaire à celle qu'on a calculée dans le cas du Réservoir uniforme avec deux événements :

- C'est la probabilité que e_m ne soit pas prise : $1 - \frac{(k*w_{m+1})}{\sum_{i=1.....m+1} w_i}$.
- La probabilité que e_m soit prise mais que l'arête ne soit pas sélectionnée pour sortir du Réservoir : $\frac{k*w_{m+1}}{\sum_{i=1.....m+1} w_i} * \frac{(k-1)}{k}$.

$$i \in [1.....m], Prob[e_i \in R_{m+1}] = Prob[e_i \in R_m] * [1 - \frac{(k * w_{m+1})}{\sum_{i=1.....m+1} w_i} + \frac{k * w_{m+1}}{\sum_{i=1.....m+1} w_i} * \frac{(k-1)}{k}]$$

$$\frac{k * w_i}{\sum_{i=1.....m} w_i} * [\frac{\sum_{i=1.....m+1} w_i - k * w_{m+1} + k * w_{m+1} - w_{m+1}}{\sum_{i=1.....m+1} w_i}]$$

$$=$$

$$\frac{k * w_i * \sum_{i=1.....m} w_i}{\sum_{i=1.....m} w_i * \sum_{i=1.....m+1} w_i} = \frac{k * w_i}{\sum_{i=1.....m+1} w_i}$$

□

2.1.3 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 présenté par Aggarwal [2] fournit un biais exponentiel en faveur des éléments récemment observés.

La fonction de biais est donnée par $f(i,t)$ où i est le ième élément de flux et t est un horodatage futur.

$$f(i,t) = e^{-\lambda(t-i)} \tag{2.1}$$

Le paramètre λ définit le taux du biais, il est généralement compris entre $[0,1]$

2.2 Graphes

Les définitions qui sont présentées dans cette section suivent largement celles de Biggs [9] et de Bollobás [12].

2.2.1 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.

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 :

$$\sum_{u \in V} d(u) = 2\,|E| \tag{2.2}$$

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 divisées par le nombre d'arêtes possibles. La densité d'un graphe non orienté est donc égale à :

$$D = \frac{2\,|E|}{|V| \cdot (|V| - 1)} \tag{2.3}$$

Pour un graphe orienté, la densité est égale à :

$$D = \frac{|E|}{|V| * (|V| - 1)} \tag{2.4}$$

Notons que le nombre maximal d'arêtes d'un graphe non orienté est :

$$\frac{|V| * (|V| - 1)}{2} \tag{2.5}$$

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

$$\frac{2\,|E|}{|V|}. \tag{2.6}$$

Demi-arêtes. Étant donné un graphe symétrique $G = (V, E)$ et une arête $e \in E$, on considère une opération de *coupure* qui définit $G'_e = (V \cup V', E')$, obtenu par la transformation suivante. Si $e = (v_1, v_2) \in E$, on génère deux nouveaux noeuds v'_1, v'_2 dans V' et on remplace l'arête e par les deux arêtes symétriques (v_1, v'_1) et (v_2, v'_2) de E'.

$G'_e = (V \cup V', E')$ est défini par $V' = \{v'_1, v'_2\}$ et $E' = E \cup \{(v_1, v'_1), (v_2, v'_2)\} - \{(v_1, v_2)\}$. Une *demi-arête* est une arête entre un noeud de V et un noeud de V'. L'opération inverse consiste à reconnecter les demi-arêtes pour obtenir G à partir de G'_e.

2.3 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. Plusieurs modèles de graphes aléatoires possibles, dans ce chapitre on va mentionner que trois modèles et finalement on va utiliser le modèle Erdös-Renyi et le modèle de configuration.

2.3.1 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.

2.3.2 Modèle Erdös-Renyi

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

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 probabilité p tel que $0 \leq p \leq 1$.*

Ce modèle est généralement appelé le modèle des graphes aléatoires Erdös-Renyi (ER), décrit par Erdös-Renyi dans deux documents de 1959 [27] et de 1960 [28].

2.3.3 Modèle de configuration

Les graphes de type Erdös-Renyi ont une distribution de degrés de type gaussienne. Dans des nombreux cas pratiques, par exemple dans les graphes issus des réseaux sociaux, les graphes ont une distribution de degré qui n'est pas gaussienne mais qui suit une loi de puissance. Nous introduisons donc le modèle de configuration qui est un modèle de graphes aléatoires qui intègre une distribution arbitraire de degrés.

Une distribution de degré $\mathcal{D}$, est une séquence qui pour un n fixe détermine le nombre de nœuds de degré 1, de degré 2, de degré i tel que la somme de ces nombres est n. Étant donné cette séquence, le nombre d'arêtes m d'un graphe avec n noeuds est la somme des degrés divisée par 2.

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

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

$$p_{ij} = \frac{d_i \cdot d_j}{2m} \tag{2.7}$$

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 (demi-arêtes) attachées aux sommets, nous pouvons en déduire que chaque matching de ce type se produit avec une probabilité égale. Ce processus peut générer des multi-graphes, c'est-à-dire des graphes avec plusieurs arêtes parallèles. Cependant tous les graphes simples sont équiprobables. La figure 2.3 illustre le processus.

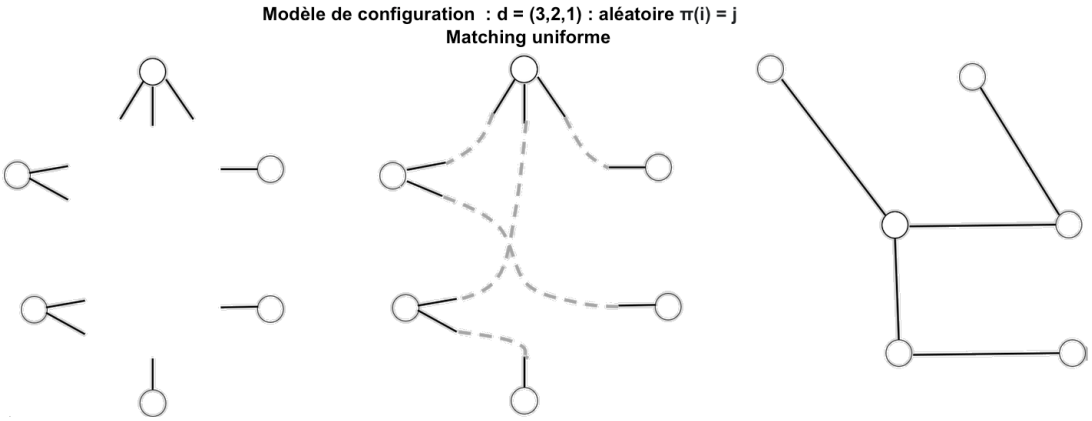

FIGURE 2.3: Exemple du modèle de configuration

2.3.4 Composante géante

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

Definition 4. *Une famille de graphe* $\{G_n : \quad n \in N\}$ *admet une composante géante si :*

$$\exists\ \alpha, n_0 \quad \forall n \geq n_0 \quad \exists \quad une \quad composante \quad connexe \quad C : \quad |C| \geq \alpha \cdot n$$

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 la transition est brutale. Pour tout ϵ, si $p = \frac{1}{n} - \epsilon$ on n'a pas de composante géante, alors que pour $p = \frac{1}{n} + \epsilon$ on a une composante géante. Ce résultat est vrai avec grande probabilité. Quand la transition est brutale, on dit qu'on a *une transition de phase*.

Le modèle Erdös-Renyi prend un graphe complet et applique des tirages sur chacune des arêtes. Si on applique Erdös-Renyi non pas au graphe complet mais sur une γ*-clique*, on peut observer que la probabilité de transition est $\frac{1}{\gamma * n}$, démontré dans [42].

Lemme 2. *Si* G *est une* γ-clique *alors pour le modèle Erdös-Renyi, il existe une composante géante si* $p \geq \frac{1}{\gamma \cdot n}$

Dans la figure 5.2, le nombre de noeuds $n = 100$, la taille de la composante géante $|C| = 40$. Il suffit de prendre $\alpha = 0,4$.

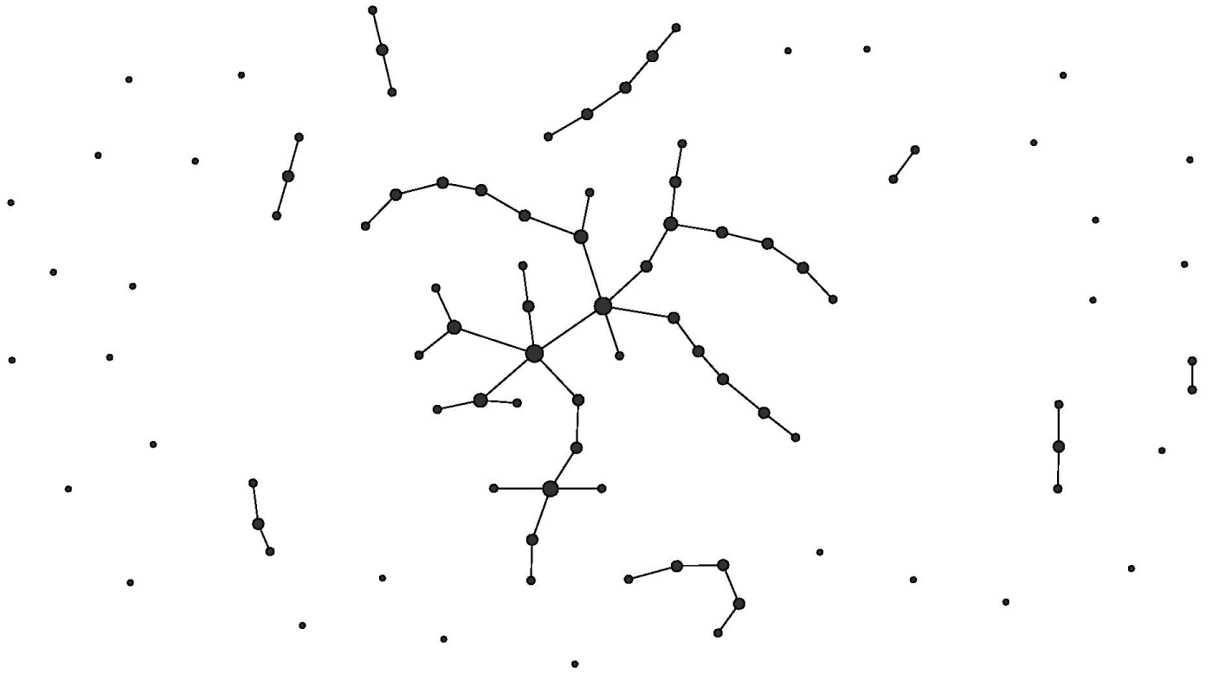

FIGURE 2.4: Exemple de Composante géante

Dans le modèle de configuration, nous cherchons une condition similaire pour observer une composante géante. Un résultat fondamental des travaux de Molloy-Reed [48] donne les conditions suffisantes pour l'existence de composante géante dans le modèle de configuration pour une distribution de degré D. Parmi ces conditions il faut que $\mathbb{E}[D^2]-2\mathbb{E}[D] > 0$ pour avoir une composante géante.

De nombreuses études dans le monde de la théorie des graphes se concentrent sur l'existence de composantes géantes ; ces recherches visent à trouver la transition de phase, en particulier lorsque nous avons une distribution arbitraire de degrés D. Dans notre cas, nous avons un Réservoir d'arêtes obtenu par la combinaison des deux modèles : nous avons un graphe G qui suit une distribution de degrés fixe (la loi de puissance), puis nous appliquons Erdös-Renyi pour obtenir un Réservoir. Sous quelles conditions, observons nous des composantes géantes dans le Réservoir ?

2.3.4.1 2-Core

Une approche classique de théorie des graphes est de construire un 2-Core à partir d'un graphe connexe. On obtient un graphe $G1$, si on répète l'élimination des noeuds de degré 1 de $G1$, on obtient un nouveau graphe $G2$. Pour obtenir le 2-Core d'un graphe, on élimine tous les noeuds de degré 1 et on répète cette opération jusqu'à obtenir des noeuds de degré au moins 2. On peut généraliser cette notion au d-Core pour $d > 2$.

Nous allons utiliser la notion de 2-Core dans la section 9.3 pour approximer une composante géante. La figure 2.5 montre un graphe et son 2-Core est spécifié par le cercle, en éliminant les noeuds $7, 8.9$. À la première itération on élimine $7, 9$, à la deuxième itération on élimine 8.

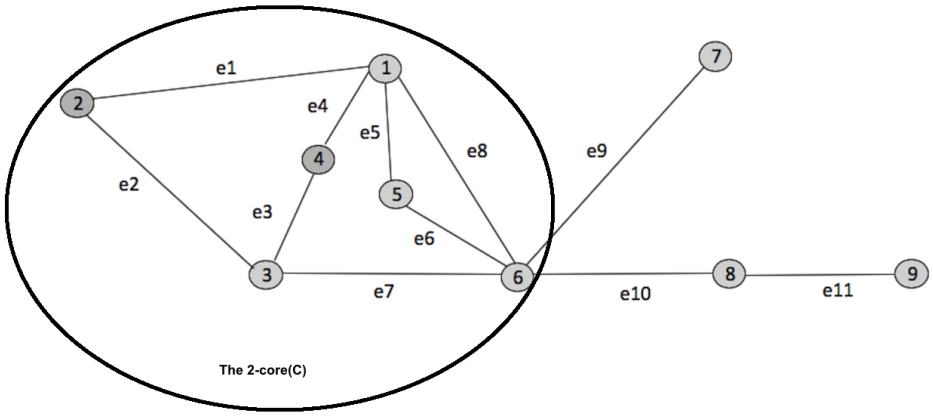

FIGURE 2.5: 2-Core d'une composante géante.

2.3.5 Conclusion

Dans ce chapitre, nous avons présenté différentes techniques d'échantillonnage pour les types de données considérés dans les chapitres suivants. Nous avons mentionné trois modèles de graphes aléatoires. Nous allons utiliser le Modèle Erdös-Renyi et le modèle de configuration.

Le *Réservoir* d'arêtes combine les deux modèles. Nous avons ensuite introduit la notion de composante géante et nous souhaitons savoir s'il existe des composantes géantes dans le Réservoir.

3 Préliminaires d'analyse du texte

Dans ce chapitre, on présente des techniques classiques d'analyse du texte :

- Le prétraitement qui va utiliser la décomposition grammaticale puis ensuite une reconnaissance d'entité.
- La représentation de mots par vecteur de petite dimension *Word2vec*.
- La classification de documents.

Le prétraitement est une opération essentielle, basée sur l'apprentissage des entités. La représentation d'un mot par vecteur suit l'analyse de la corrélation des occurrences de deux mots dans la même phrase, ou le deuxième moment de la distribution des mots [1].

Étant donnée une matrice de corrélation (n, n), ou toute matrice positive semi-définie symétrique, on peut associer un vecteur v_i dimension n à valeur réelle pour $i = 1, ...n$, de telle sorte que le produit scalaire $v_i.v_j$ soit égal à $A(w_i, w_j)$. Si nous ne sélectionnons que les grandes valeurs propres de A, nous pouvons obtenir des vecteurs de plus petite dimension tels que $v_i.v_j \simeq A(w_i, w_j)$.

La classification de documents est une opération classique. On va décrire deux méthodes pour réaliser des ce type de tâches.

3.1 Prétraitement des données

3.1.1 Lemmatisation et Entités Nommées

L'objectif de la lemmatisation est de réduire les formes flexionnelles et parfois les formes dérivées d'un mot à une forme de base commune, la racine, en utilisant une décomposition grammaticale. La lemmatisation consiste en une analyse morphologique des mots, visant à supprimer les terminaisons inflexibles et à trouver la forme de base d'un mot. Celle-ci est aussi appelée lemme qu'on peut aussi trouver dans un dictionnaire. La figure 3.1 montre la lemmatisation de la phrase :
When I arrived in New York, around June 10th 2018, I lived across the First National Bank of Chicago.

1. Le deuxième moment est la distribution des paires de mots, le troisième moment est la distribution des triplets de mots et le k-th moment est la distribution des k mots dans la même phrase.

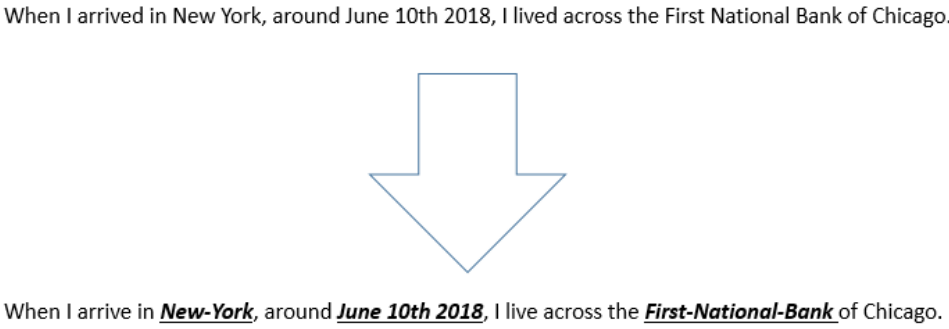

FIGURE 3.1: Phrase lemmatisée.

La reconnaissance d'entités nommées (NER) est une sous-tâche d'extraction d'informations qui cherche à localiser et à classer les entités nommées mentionnées dans un texte dans des catégories prédéfinies telles que les noms de personnes, les organisations, les lieux, etc [49]. Le NER agit non seulement comme un outil autonome d'extraction d'informations (IE), mais joue également un rôle essentiel dans diverses applications de traitement du langage naturel (NLP) telles que le *compréhension de texte* [66], les *systèmes de question-réponse* [24], la *traduction automatique* [5], etc.

La classification consiste à attribuer une étiquette telle que "nom", "lieu", "date", "email", etc. Si le NER désigne au départ l'extraction de noms propres, de noms de lieux et de noms d'organisations, ce concept s'est étendu à d'autres entités telles que la date, l'email, le montant d'argent, etc.

Il existe trois grandes approches des NER :

- basée sur un lexique
- basée sur des règles grammaticales
- basée sur l'apprentissage machine.

Un système (NER) peut combiner plusieurs de ces techniques [37]. L'approche grammaticale est utilisée dans l'étiquetage morphosyntaxique où Part-of-Speech (POS) Tagging ou on essaye d'attribuer une étiquette à chaque mot d'une phrase mentionnant la fonctionnalité grammaticale d'un mot (Nom propre, adjectif, déterminant...).

Le tableau 3.1 présente les principaux outils utilisés pour le marquage des (NER), le *SpaCy* et le *NLTK* sont basés sur des approches statistiques. Alors que le NER du *CoreNLP* NER de Stanford utilise un processus d'apprentissage à l'aide d'un réseau de neurones qui nécessite une grande masse de données.

Prenons l'exemple suivant : *" When I arrived in New York, around June 10th 2018, I lived across the First National Bank of Chicago."*. La figure 3.2 montre la classification

Library Name
SpaCy
CoreNLP
NLTK

TABLE 3.1: Outils pour la reconnaissance des entités

des entités nommées de cet exemple avec l'outil de Stanford et l'utilisation de la librairie SpaCy.

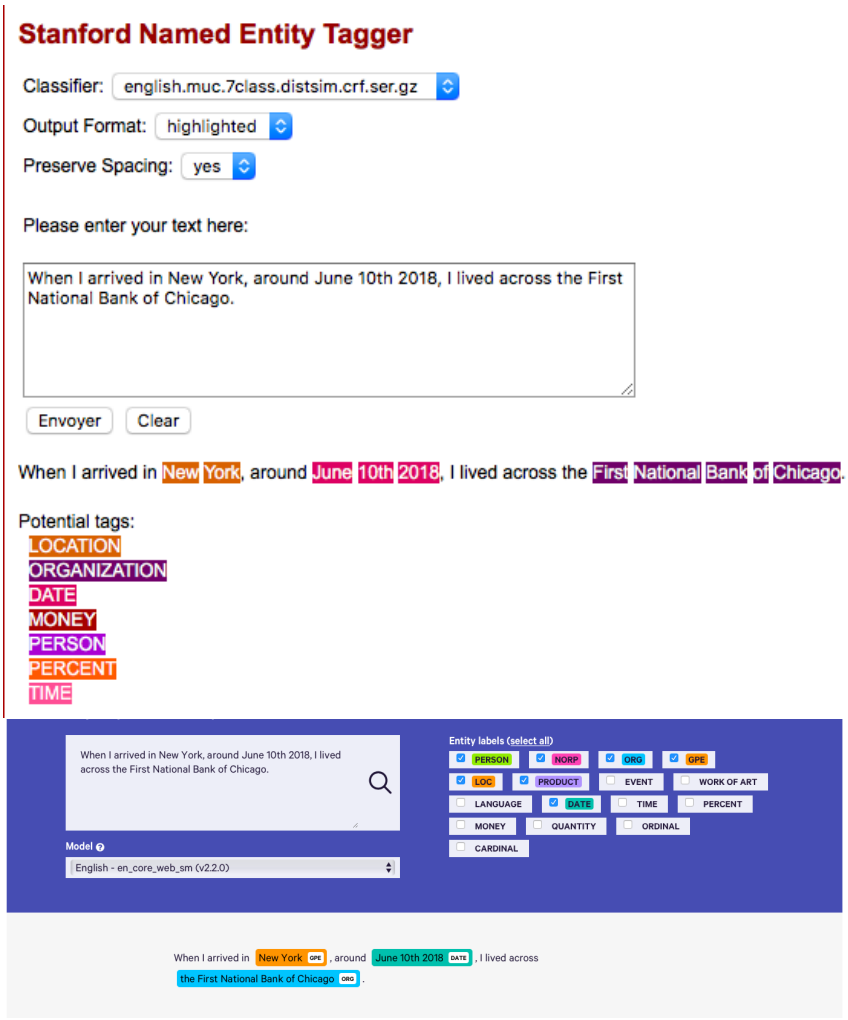

FIGURE 3.2: Outil NER de Stanford avec la librairie SpaCy.

Ces opérations ne sont pas parfaites : en particulier certains systèmes vont reconnaitre *National Bank of Chicago* comme entité et considérer *First* comme un adjectif.

3.1.1.1 Décomposition grammaticale classique

Une des représentations grammaticales classiques est celle des grammaires hors-contexte (CFG, Context-Free Grammars) largement utilisée pour la description des langages de programmation.

La figure 3.3 montre une grammaire hors-contexte très simple. Dans ce cas, l'ensemble de non-terminaux spécifie quelques catégories syntaxiques de base : par exemple S signifie "phrase", NP "expression nominale", VP "expression verbale", etc. Σ contient l'ensemble

N = {S, NP, VP, PP, DT, Vi, Vt, NN, IN}
S = S
Σ = {sleeps, saw, man, woman, dog, telescope, the, with, in}

R =

S	→	NP	VP
VP	→	Vi	
VP	→	Vt	NP
VP	→	VP	PP
NP	→	DT	NN
NP	→	NP	PP
PP	→	IN	NP

Vi	→	sleeps
Vt	→	saw
NN	→	man
NN	→	woman
NN	→	telescope
NN	→	dog
DT	→	the
IN	→	with
IN	→	in

FIGURE 3.3: Un simple exemple d'une Grammaires hors-contexte, l'ensemble des non-terminaux N contient un ensemble de base de catégories syntaxiques : S = Sentence, VP = Verb Phrase, NP = Noun Phrase, PP = Prepositional Phrase, DT = Determiner, Vi = Intransitive Verb, Vt = Transitive Verb, NN = Noun, IN = Preposition. Σ est l'ensemble des mots possibles dans la langue.

des mots du vocabulaire. Le symbole de départ dans cette grammaire est S : cela spécifie que chaque arbre d'analyse a S comme racine.

Une décomposition est un arbre syntaxique qui est le témoin du fait que la phrase est grammaticalement correcte. Dans le cas d'une grammaire context-free on l'obtient avec l'algorithme CKY(Cocke–Kasami–Younger) [36] qui permet de trouver l'arbre de décomposition en $O(n^3)$ en utilisant la programmation dynamique.

3.1.1.2 La décomposition par dépendances

La décomposition par dépendances [57] procède différemment de l'algorithme CKY et décompose la phrase en deux parties : la partie gauche qui est la pile, la partie droite qui est le buffer. Le but de la décomposition par dépendances est de trouver un arbre de décomposition. À chaque étape, l'algorithme de décomposition lit un nouveau mot et doit décider une des trois actions possibles :

- Empiler le mot.
- Trouver les dépendances [57] à l'intérieur des mots de la pile .

- Certains mots vont devenir des mots importants qu'on va rattacher à la racine, par-exemple les verbes.

L'algorithme de décomposition apprend à partir d'exemples l'action de décomposition pour un mot donné. Un exemple est un arbre de décomposition associé à une phrase avec des différentes dépendances, comme l'indique le treebanks de la figure 3.4. Le treebank décrit des milliers d'exemples de cette nature.

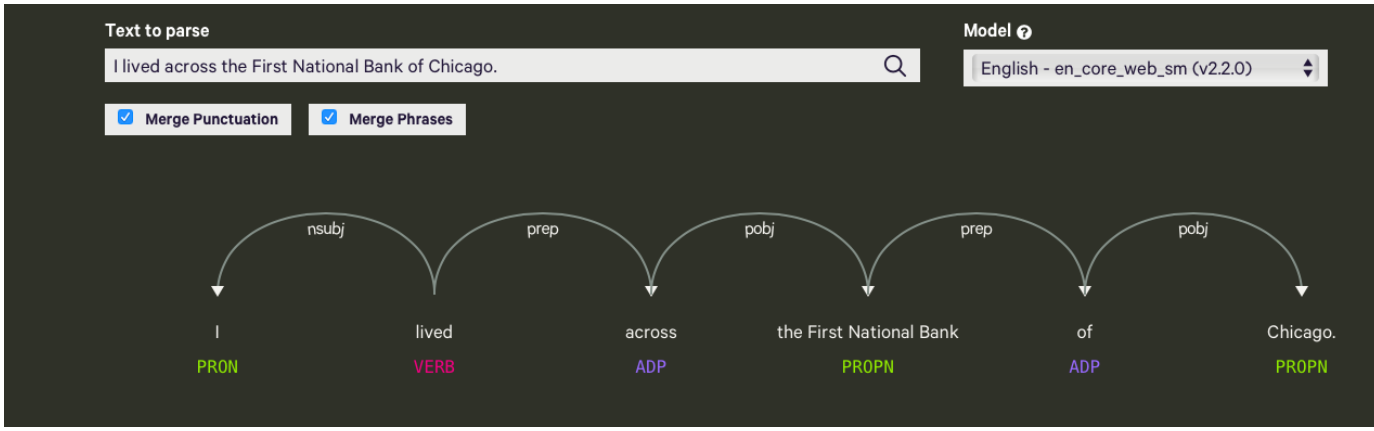

FIGURE 3.4: Treebank

3.2 Vectorisations des mots : *Word2vec*

Les représentations vectorielles des mots (*word embeddings*) essaient de saisir les relations entre les mots sous forme de distance ou d'angle, et ont de nombreuses applications en traitement du langage naturel et en apprentissage machine. Les méthodes les plus simples utilisent des vecteurs de mots obtenus directement à partir de la matrice de co-ocurrence. Les heuristiques de repondération sont connues pour améliorer ces méthodes, tout comme la réduction de dimension [23]. Certaines méthodes de repondération sont non linéaires, notamment la prise en compte de la racine carrée des nombres de co-ocurrence [53], ou le logarithme, ou les informations ponctuelles *PMI* (Pointwise Mutual Information) qui s'y rapportent [20].

Les réseaux neuronaux [[55], [54], [7], [21]] proposent une autre façon de construire des *embeddings*. Le vecteur associé à un mot est simplement la représentation interne du réseau neuronal lorsque l'entrée est le mot donné. Cette méthode est non linéaire et non convexe. Elle a été popularisée par *Word2vec*, une famille de modèles basés sur l'énergie [[44], [45], [46]], et par une approche de factorisation de matrice appelée *GloVe* [51]. Le premier article a également montré comment résoudre des analogies en utilisant l'algèbre linéaire sur les *embeddings*. Les expériences et la théorie ont été utilisées pour suggérer

que ces nouvelles méthodes sont liées aux anciens modèles basés sur le PMI, mais avec de nouveaux hyperparamètres et/ou des méthodes de repondération des termes [39].

Les chercheurs en traitement du langage utilisent des approximations de faible dimension. Les méthodes *Word2vec* et *Glove* au départ, sont des méthodes expérimentales. L'article [4] fournit une justification théorique aux modèles non linéaires comme *PMI, Word2vec et GloVe*, ainsi qu'à certains choix d'hyperparamètres.

Étant donnée une matrice de corrélation (n, n), ou toute matrice positive semi-définie symétrique, on peut associer un vecteur v_i de dimension n pour $i = 1, ...n$, de telle sorte que le produit scalaire $v_i.v_j$ est égal à $A(w_i, w_j)$. Si nous ne sélectionnons que les grandes valeurs propres de A, nous pouvons obtenir des vecteurs de plus petite dimension tels que $v_i.v_j \simeq A(w_i, w_j)$. On peut considérer les coefficients de la matrice de corrélation comme les probabilités d'observer une paire de mots dans une phrase tirée aléatoirement dans ce document puis ensuite en prenant deux mots tirés aléatoirement dans cette phrase. Les valeurs diagonales de la matrice donnent le premier moment de cette distribution.

Pour $n = 10^4$, cette méthode n'est pas réaliste et on doit utiliser des méthodes plus efficaces telles que *Word2vec*, qui est utilisée pour obtenir des vecteurs de dimension 300 lorsque le dictionnaire comporte $n = 10^4$ mots, c'est-à-dire réduire la dimension. Considérons un corpus de 10000 mots, et dans ce corpus les deux mots suivants (ants, abandon). Chaque mot est codé par son vecteur one-hot [2](ici des matrices de colonnes de 10 000 lignes) comme le montre la figure 3.5.

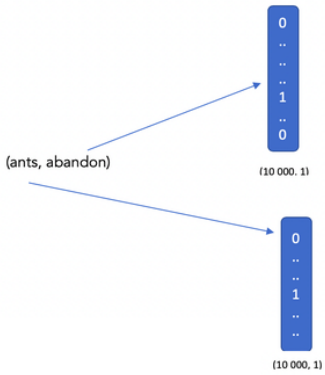

FIGURE 3.5: Vecteur one hot

Un réseau de neurones récurrents RNN observe des fenêtres de M mots dans une fenêtre coulissante. Les variables latentes du réseau définissent la matrice de la figure 3.6.

2.

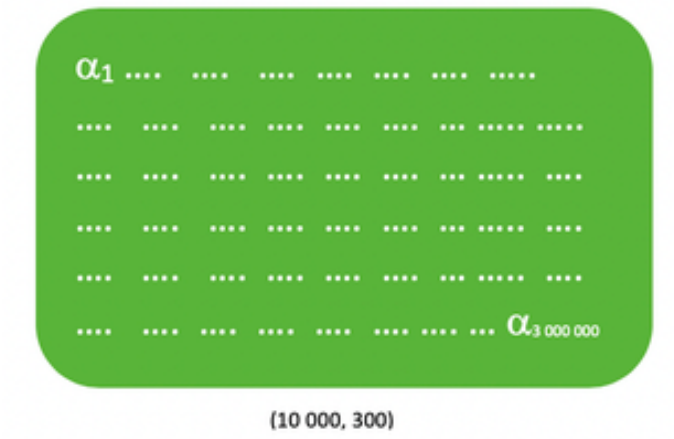

FIGURE 3.6: Matrice 10 000 lignes et 300 colonnes

Cependant, après la transformation du couple de mots en un vecteur one hot, la figure 3.7 montre comment la couche cachée sera transformée. La similarité entre deux mots est

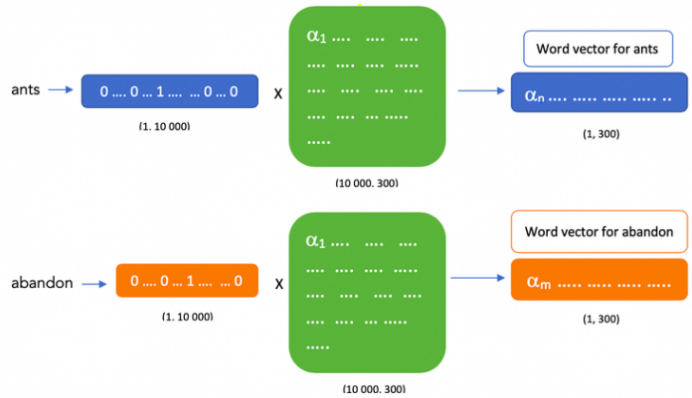

FIGURE 3.7: Matrice 10 000 lignes et 300 colonnes

le produit scalaire des vecteurs de cette matrice de la figure 3.8.

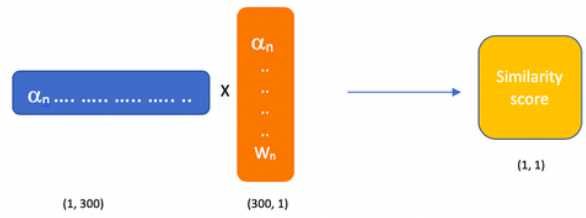

FIGURE 3.8: Vecteur de similarité

3.3 Classification

La classification de documents consiste à définir un nombre fixe de classes, en général un paramètre k de manière que chaque document soit proche d'une des classes.

3.3.1 IRaMuTeQ

IRaMuTeQ [52](Interface de R pour les Analyses Multidimensionnelles de Textes et de Questionnaires) présente des analyses statistiques des matrices associées à un ensemble de documents. Soit M la matrice avec i lignes, où i est l'index du i-ème documents et j colonnes où j est l'index du j-ème mots, $M(i, j)$ est le nombre d'occurrences du mot j dans le document i. IRaMuTeQ permet de faire différents types d'analyse : statistiques textuelles classiques, spécificités des groupes, classification hiérarchique descendante, analyse de similarité de mots et de documents. La méthode IRaMuTeQ [52] permet des classifications hiérarchiques basées sur l'analyse en composantes principales (PCA) qui remonte aux années 60s. En pratique on peut utiliser la décomposition SVD (Singular Value Decomposition) d'une matrice (n, n) qui a cependant une complexité temporelle en $O(n^3)$.

L'inconvénient de cette méthode est le passage à l'échelle quand n est très grand. Pour l'analyse de telles données, d'autres méthodes sont apparues telles que LDA.

3.3.2 LDA

Un modèle thématique, par exemple LDA (Latent Dirichlet Allocation) [10] apprend une représentation de faible dimension des documents comme un mélange de variables latentes appelées sujets (*Topics*). Les sujets sont des distributions multinomiales sur un vocabulaire prédéfini de mots. Puisque les mots sont observés, les distributions des documents et des sujets, respectivement θ et ϕ sont conditionnellement indépendantes. En outre, les documents ne sont pas directement liés aux mots. Cette relation est plutôt régie

Symbol	DESCRIPTION
D	total number of documents
k	number of topics
V	total number of unique words
w	the unique word associated with the i th token
Z	the topic associated with w

TABLE 3.2: Notation utilisée dans le cadre de LDA

par des variables latentes supplémentaires, z , introduites pour représenter la responsabilité d'un sujet particulier dans l'utilisation de ce mot dans le document, c'est-à-dire le(s) sujet(s) sur lequel (lesquels) le document est focalisé. Un glossaire des notations utilisées dans le document est donné dans le tableau 3.2.

Pour compléter son processus de génération des documents, le modèle *LDA* considère pour les distributions de documents sur les sujets et les distributions de sujets sur les mots des distributions de *Dirichlet priors sparse* c'est-à-dire chaque document a peu de topics et le support de distribution est petit. Le *LDA* est basé sur l'hypothèse de l'échangeabilité des mots dans un document et des documents dans un corpus.

Dans le cadre de LDA, les documents sont supposés être générés de manière aléatoire à partir d'un ou plusieurs sujets, chacun d'entre eux étant une distribution de mots. Les thèmes sont considérés comme des variables latentes, et LDA s'exécute en déduisant les thèmes des documents par un processus de *Dirichlet*. L'algorithme échantillonne les documents à plusieurs reprises et modifie les thèmes pour mieux les adapter jusqu'à atteindre une convergence spécifiée.

3.4 Mécanismes d'Attention

Les mécanismes d'attention [40, 60] prévoient pour un mot w_i dans une phrase, la distribution des mots les plus corrélés. La corrélation d'un mot w_j avec w_i est approximativement la valeur $v(w_i).v(w_j)$. On peut alors calculer $\{v(w_i).v(w_j) : j \neq i\}$ pour un mot w_i fixe et normaliser les valeurs pour obtenir une distribution. Ce mécanisme est principalement utilisé dans les transformateurs [63] pour la traduction automatique. Nous montrons dans la section 6.5 que le Réservoir pondéré peut reconstruire les distributions d'attention les plus significatives. Nous définissons les *distributions les plus significatives* comme les distributions des nœuds centraux dans un cluster.

L'analyse de l'attention de la phrase :
On September 11th, 2001, the attack on the World Trade Center had a huge impact on American politics.
pour les entités *September 11th, 2001* et *the World Trade Center* est donnée dans la figure 3.9.

3.4.1 Conclusion

Dans ce chapitre, nous avons présenté des techniques utilisées pour le prétraitement de données que nous allons ensuite utiliser comme des boites noires. Nous utiliserons la méthode *Word2vec* dans le chapitre **??**, pour analyser des données de texte en streaming. Nous avons présenté deux méthodes classiques de classification pour les comparer avec notre méthode dans le chapitre **??**. Les mécanismes d'Attention sont directement liés aux méthodes d'échantillonnages que nous allons utiliser.

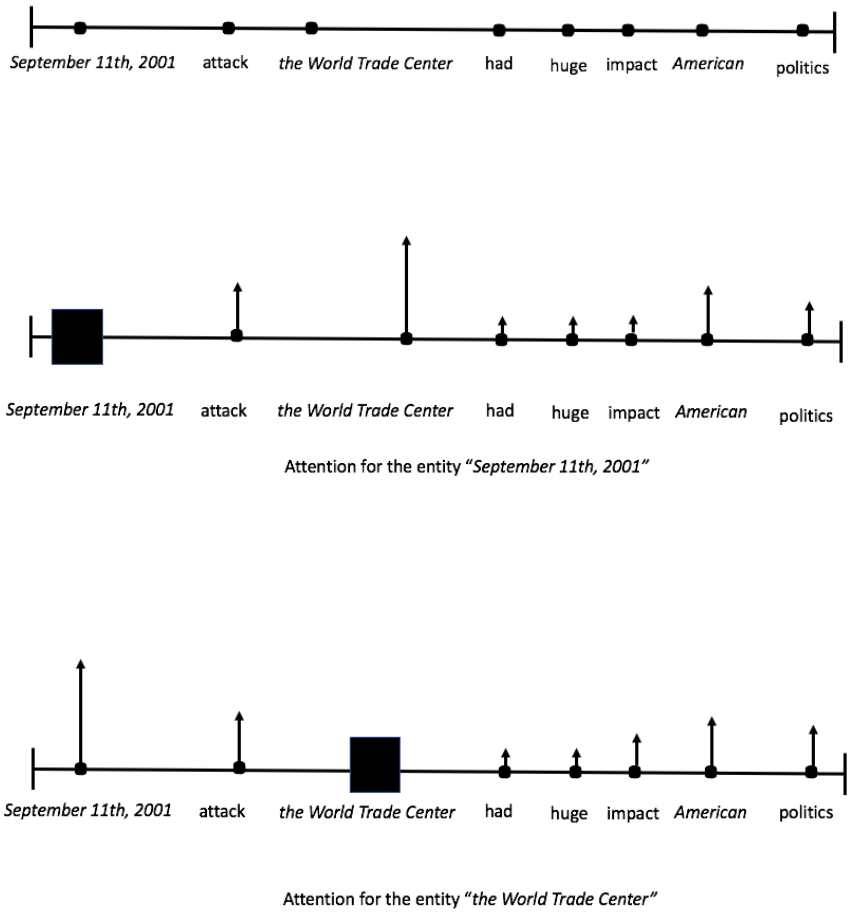

FIGURE 3.9: L'attention pour un mot spécifique.

4 Analyse OLAP sur des flux de données

Un entrepôt de données selon Bill Inmon [35], est une collection non volatile et variable dans le temps, de données orientées vers un sujet et intégrées à partir de sources de données. Chacun des termes est défini comme suit :

- Orienté sujet. Dans l'entrepôt de données, les données sont organisées par thème. Les données propres à un thème, les ventes par exemple, sont rapatriées de différentes bases.
- Intégré. Les données proviennent de sources hétérogènes utilisant chacune un type de format. Elles sont modifiées avant d'être intégrées dans l'entrepôt.
- Non volatile. Les données ne disparaissent pas et ne changent pas au fil des traitements, au fil du temps.
- Variable dans le temps. Les données non volatiles sont horodatées. On peut ainsi visualiser l'évolution dans le temps d'une valeur donnée. Le degré de détail de l'archivage est relatif à la nature des données. Toutes les données ne méritent pas d'être archivées.

On peut considérer l'entrepôt de données avec deux points de vue différents :

- Un flux de tuples $t_1, t_2, ..., t_n, ...$, et on ne stocke pas tous les tuples.
- On stocke tous les tuples, pour former une relation.

Si on suit la première approche, on s'intéressera à l'approximation des requêtes d'analyse et le résultat ne sera qu'approché. Si on suit la deuxième approche, l'entrepôt de données est une table d'une base de données qui suit un modèle relationnel et permet de réaliser des analyses. L'entrepôt de données est alors considéré comme une table où on peut seulement ajouter des tuples.

Un schéma d'analyse ou un schéma OLAP spécifie des dimensions d'analyse et des *Mesures* associées aux tuples de l'entrepôt.

Dans l'analyse OLAP classique, nous stockons l'ensemble de l'entrepôt de données et une requête OLAP est définie par un filtre, d dimensions et une *Mesure* parmi un ensemble fini des *Mesures* $M_1....M_l$. Les *préférences* sont des nouvelles *Mesures* considérées comme une distribution sur $M_1....M_l$.

La première section introduit les principaux concepts : entrepôts de données, schémas OLAP. La deuxième section détaille l'approximation des requêtes OLAP en streaming. La troisième section étudie la gestion des préférences dans ce cadre.

4.1 Entrepôts de données et requêtes OLAP

Considérons un schéma relationnel avec trois relations (Achat,Clients,Produits) comme indiqué sur la figure 4.1. Un schéma OLAP 4.2 ou schéma étoile pour ce schéma relationnel, spécifie la table Achat comme table d'analyse et décrit un arbre où chaque nœud est étiqueté soit par un nom de table, soit par un ensemble d'attributs. La racine est étiquetée par la table d'analyse qui peut aussi être représentée par la liste de tous ses attributs. Une arête de l'arbre existe s'il existe une dépendance fonctionnelle entre les attributs du nœud d'origine et les attributs du nœud d'extrémité pour les tables du schéma relationnel 4.1. Les *Mesures* sont des attributs numériques de la table Achat. Dans l'exemple de la figure 4.1, les trois attributs numériques sont : *Speed, Quantity, Price*.

Achat(ID_c, ID_P, Day, Speed, Quantity, Price)
Clients(ID_c, name, age, gender, csp)
Produits(ID_P, article, type, artiste, price)

FIGURE 4.1: Schéma relationnel

Un Entrepôt de données, noté I, est une table particulière stockant des tuples t avec les attributs $A_1, ...A_m, M_1...M_l$, certains A_i étant des clés étrangères d'autres tables, et $M_1....M_l$ les *Mesures*. Des tables de dimensions fournissent d'autres attributs liés à la table d'analyse par des clés.

Une requête OLAP pour un schéma S est déterminée par :

- Une condition de filtre qui sélectionne un sous-ensemble des tuples de l'entrepôt.
- Une *Mesure* et un opérateur d'agrégation (COUNT, SUM, AVG, ...).
- d dimensions qui sont représentées par des nœuds de l'arbre. Une dimension peut aussi être appelée dans certains contextes comme un classificateur.

Une *Mesure* est un attribut numérique, et on considère dans la suite l'opérateur d'agrégation comme la somme (SUM). La réponse à une requête Q est un tableau multidimensionnel, selon les dimensions $C_1, ...C_d$ et la *Mesure* $t.M$. Chaque tuple $c_1, ..., c_d, m$ de la réponse, où $c_i \in C_i$, est tel que :

$$m = \frac{\sum_{t:t.C_1=c_1,...t.C_d=c_d} t.M}{\sum_{t \in I} t.M}.$$

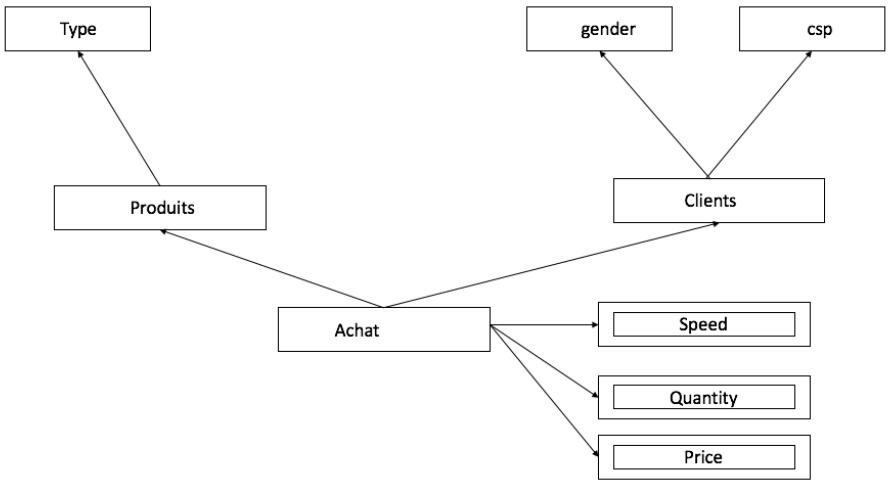

FIGURE 4.2: Schéma OLAP.

Nous considérons m, la valeur relative de la réponse alors que le numérateur est la valeur absolue de la réponse et nous écrivons Q_C^I comme la distribution ou *vecteur de densité* pour la réponse à Q sur les dimensions $C_1, ...C_d$ et sur l'entrepôt de données I. Prenons par exemple Q_{gender} et $Q_{gender,type}$ comme des requêtes sans filtre, la dimension gender pour la première et (gender, type) pour la deuxième, la *Mesure* Prix et l'agrégation SUM.

La réponse à la requête Q est indiquée à la figure 4.3, considérée comme une distribution. *Gender* est un attribut dont les valeurs possibles sont {*homme, femme*}, et *type* est un attribut dont les valeurs possibles sont {*pop, rock,classique*}.

4.2 Approximation à partir d'échantillons

Dans notre contexte, nous approximons une distribution Q. Il y a généralement deux paramètres $0 \leq \varepsilon, \delta \leq 1$ pour l'approximation des algorithmes randomisés, où ε est l'erreur, et $1-\delta$ la confiance. Considérons la réponse Q à une requête OLAP comme une distribution sur un support fini, et supposons qu'un algorithme probabiliste A calcule $\widehat{Q}$. La norme L_1 appliquée à $|Q - \widehat{Q}|_1 = \sum_i |Q[i] - \widehat{Q}[i]|$. L'algorithme (ε, δ)-approxime Q si :

$$Prob_\Omega[|Q - \widehat{Q}| \leq \varepsilon] \geq 1 - \delta$$

L'espace probabiliste Ω est déterminé par les choix de l'algorithme A. Une approche classique consiste à prendre k échantillons aléatoires, où la probabilité d'un échantillon t est proportionnelle à sa *Mesure* $t.M$. La réponse $\widehat{Q}[i]$ est la proportion d'échantillons où la valeur de l'attribut est i. Elle peut également être considérée comme la même requête

FIGURE 4.3: Une requête OLAP de dimension 1 et une requête de dimension deux.

OLAP sur un nouveau entrepôt de données construit avec k échantillons où la *Mesure* originale M est remplacée par la constante 1.

Pour un entrepôt de données I, soit $\widehat{I_M}$ un ensemble d'échantillons obtenus à l'aide de l'échantillonnage pondéré sur une *Mesure* M, comme représenté dans la figure 4.5.

Algorithme 3 Algorithme pour calculer $\widehat{Q_C}$

Entrée : I
- Construire $\widehat{I_M}$,
- Pour toute valeur $c(C)$, la valeur $\widehat{Q}_{C=c}$ est le nombre de tuples $\widehat{I_M}$ tels que $C = c$
- $\widehat{Q_C}$ est la distribution ainsi obtenue pour tout c.

Remarquons que $\widehat{Q_C}$ est la réponse à la même requête OLAP sur $\widehat{I_M}$, où la nouvelle mesure est une constante 1.

Soit $\mid C \mid$ le nombre de valeurs possibles c de l'attribut C, c'est-à-dire le support de C que nous supposons fini, et soit $1_{C=c}(t)$ la fonction caractéristique d'un tuple t pour le critère $C = c$,

$$1_{C=c}(t) = \begin{cases} 1 & \text{si } \ t.C = c \\ 0 & \text{sinon.} \end{cases}$$

Par hypothèse, le Réservoir R de taille k est exactement $\widehat{I_M}$ et

$$Prob[t \in \widehat{I_M}] = \frac{k \cdot t.M}{T}$$

comme on a vu dans l'étude du chapitre 2.1.2.

Comme il a été montré dans [22], $\widehat{Q}_C$ est une bonne approximation de Q_C. Pour chaque valeur possible c de l'attribut C, l'espérance de $\widehat{Q}_{C=c}$ est le nombre attendu de tuples t tel que $t.C = c$, est précisement $Q_{C=c}$.

Lemme 3. *L'espérance de $\widehat{Q}$, écrite $\mathbb{E}(\widehat{Q})$ est la valeur exacte $Q_{C=c}$.*

Démonstration. Supposons que nous ayons qu'une seule dimension C, pour simplifier. Considérons la densité pour $C = c$ et son espérance :

$$\mathbb{E}(\widehat{Q}_{C=c}) = \frac{\sum_t Prob[t \in \widehat{I_M}] \cdot 1_{C=c}(t)}{k} =$$

$$\frac{\sum_{t:\ t.C=c} Prob[t \in \widehat{I_M}]}{k} =$$

$$\frac{\sum_{t:\ t.C=c} \frac{k.t.M}{T}}{k} = \frac{\sum_{t:\ t.C=c} t.M}{T} = Q_{C=c}$$

□

Pour la distribution Q_C, comme vecteur, nous avons $\mathbb{E}(\widehat{Q}_C) = Q_C$. L'un des objectifs est d'obtenir un algorithme d'approximation A qui produit une distribution $A(Q_C) = \widehat{Q}_C$ de telle sorte que

$$Prob[|\ Q_C - \widehat{Q}_C)\ |_1 \leq \varepsilon] \geq 1 - \delta$$

où nous prenons la norme L_1 entre les deux vecteurs. Elle peut être obtenue si nous prenons plusieurs Réservoirs indépendants et prenons les valeurs moyennes de chaque composante. La borne de *Chernoff-Hoeffding* [33] permettra d'obtenir l'inégalité ci-dessus.

Soit $X_1, ..., X_n$ des variables aléatoires indépendantes délimitées par l'intervalle : $0 \leq X_i \leq 1$. La moyenne empirique de ces variables est :

$$\overline{X} = \frac{1}{n}(X_1 + \cdots + X_n).$$

L'une des inégalités de *Hoeffding* est :

$$\mathrm{P}\left(\overline{X} - \mathrm{E}\left[\overline{X}\right] \geq t\right) \leq e^{-2nt^2}$$

La déviation à la moyenne décroit exponentiellement en fonction du nombre de tirages n.

Avec un seul Réservoir, l'algorithme A qui produit $\widehat{Q}_C$ peut aussi être analysé avec l'inégalité de Markov [1], mais l'approximation n'est pas aussi bonne :

Lemme 4. *Pour toutes les valeurs c, il existe ε, δ tel que $Prob[|\ Q_{C=c} - \widehat{Q}_{C=c})\ |_1 \leq \varepsilon] \geq 1 - \delta$.*

1. Pour $a \geq 0$, $\quad Prob[\widehat{X} \geq a \cdot \mathbb{E}(X)] \leq 1/a$

Démonstration. Pour $a \geq 1$, $Prob[\widehat{Q}_{C=c} \geq a \cdot \mathbb{E}(\widehat{Q}_{C=c}] \leq 1/a$ qui peut également être réécrite comme :

$$Prob[|\ \widehat{Q}_{C=c} - Q_{C=c})\ | \geq (a-1) \cdot Q_{C=c}] \leq 1/a$$

Nous fixons $\varepsilon = (a-1) \cdot Q_{C=c}$, donc $a = 1 + \varepsilon/Q_{C=c}$ et $\delta = 1/a \simeq 1 - \varepsilon/Q_{C=c}$:

$$Prob[|\ \widehat{Q}_{C=c} - Q_{C=c})\ | \geq \varepsilon] \leq \delta$$

$$Prob[|\ \widehat{Q}_{C=c} - Q_{C=c})\ | \leq \varepsilon] \geq 1 - \delta$$

□

4.2.1 Construction des échantillons à partir d'un entrepôt de données

Un des buts, est d'approximer une requête OLAP à partir des échantillons pris avec une probabilité proportionnelle à une *Mesure*, comme par exemple *Speed*, qui représente la vitesse de la transaction ou l'inverse du temps de transaction pour l'entrepôt de données Achat. Comment réaliser un tel tirage ?

Nous modifions le schéma de la table Achat en ajoutant une colonne qui représente le poids cumulatif de la *Mesure Speed*. Le nouvel attribut cumulatif est *Cumulative speed* comme le montre la figure 4.4 en partant de la valeur cumulative de 0.

Dans l'exemple de la figure 4.4, la valeur maximum de l'attribut *Cumulative speed* est de 5200. La procédure est alors la suivante :

- tirer une valeur uniforme λ entre 0 et 5200.
- trouver un tuple t tel que sa valeur cumulative est strictement plus grande ou égale à λ et le tuple précédent à une valeur cumulative plus petite que λ.

Dans ce cas, la probabilité de choisir tuple t est proportionnelle à sa *Mesure Speed*. Nous procédons par dichotomie, en comparant λ à la valeur cumulative du tuple en position $n/2$. Si λ est supérieur, on isole la partie la plus récente et on recommence. Si λ est inférieur, on sélectionne la partie la plus ancienne et on recommence.

Datawarehouse

ID_C	ID_P	Day	Speed	Cumulative speed
10	100	Monday	1000	1000
20	200	Monday	300	1300
10	300	Tuesday	800	2100
20	100	Tuesday	500	2700
30	200	Tuesday	1000	3700
10	100	Sunday	700	4400
20	200	Sunday	100	4500
30	300	Sunday	700	5200

FIGURE 4.4: Entrepôt de données.

Pour approximer une requête OLAP, il faut appliquer l'algorithme **??** décrit dans la section 4.2.

4.2.2 Approximation des requêtes OLAP sur les données de streaming

Supposons que les tuples de l'entrepôt de données arrivent comme un flux $t_1, t_2, ...t_i, ...$ Nous appliquons la méthode d'échantillonnage pondéré, détaillée dans le chapitre 2.1.2.

La principale propriété est que chaque tuple t_i est pris proportionnellement à sa *Mesure*. C'est un argument simple classique que nous rappelons dans la section 2.1.2. Une fois le Réservoir construit, on peut approximer des requêtes OLAP comme dans la section 4.2, sans stocker tous les tuples.

4.3 Préférences pour l'analyse OLAP

Pour l'analyse de données, les préférences englobent une grande variété de possibilités. Elles peuvent être définies comme des conditions logiques, telles que la géolocalisation, mais peuvent aussi être des profils globaux représentés comme une distribution sur un support fini. Un profil Netflix est une telle distribution sur 250 catégories. Il s'agit généralement d'un vecteur sparse α de dimensions 250 tel que $\sum_i \alpha_i = 1$. Une préférence générale est une combinaison de ces deux composantes : les contraintes logiques et des profils de distribution. Ces deux composantes ont des impacts très différents sur l'analyse des données.

Supposons un schéma OLAP, avec plusieurs *Mesures* M_1, M_2, M_3 comme dans la figure 4.2. Une préférence pour le schéma OLAP est une distribution sur le support M_1, M_2, M_3.

La figure 4.5 décrit l'entrepôt de données Achat avec deux clés étrangères ID_C pour la table *Clients* et ID_P pour la table *Produits*. Il y a 3 *Mesures* et une préférence est une distribution telle que :

$$M_\alpha = \alpha.\text{Speed} + \beta.\text{Quantity} + (1 - \alpha - \beta).\text{Price}$$

pour certains $0 \leq \alpha, \beta \leq 1$.

Un profil de préférence est une distribution sur les *Mesures* possibles du tuple t. Imaginons le scénario dans lequel nous avons un flux de tuples t et nous voulons analyser rapidement les données avec des requêtes OLAP et la nouvelle *Mesure* M_α. Nous construisons autant de Réservoirs que de *Mesures*. Le profil est inconnu lorsque nous prélevons les échantillons. Nous pouvons construire un nouveau Réservoir à partir des échantillons existants et l'utiliser pour approximer les requêtes OLAP pour la nouvelle *Mesure*.

Pour une taille du Réservoir k donnée et le schéma OLAP de la figure 4.2, nous construisons trois Réservoirs indépendants, R_1, R_2 et R_3 avec k échantillons pour chacune des *Mesures*. La construction suit l'échantillonnage k-pondéré, comme le montre la figure 4.5. Si une préférence M_α est définie à tout moment, *après avoir prélevé les échantillons*, nous pouvons construire un nouveau Réservoir R_α de taille k. Nous montrons que l'évaluation de la requête Q sur R_α est une approximation de Q, comme dans la section 4.2.2.

Construction d'un Réservoir R_α

- Prenons $\alpha.k$ échantillons uniformes de R_1,
- Prenons $\beta.k$ échantillons uniformes de R_2,
- Prenons $(1 - \alpha - \beta).k$ uniform samples from R_3.

Nous montrons que l'évaluation de la requête Q sur R_α est une approximation de Q. Nous supposons pour simplifier que $p = 2$. Une préférence s'écrit comme :

$$M_\alpha = \alpha.M_1 + (1 - \alpha).M_2$$

pour $0 \leq \alpha \leq 1$.

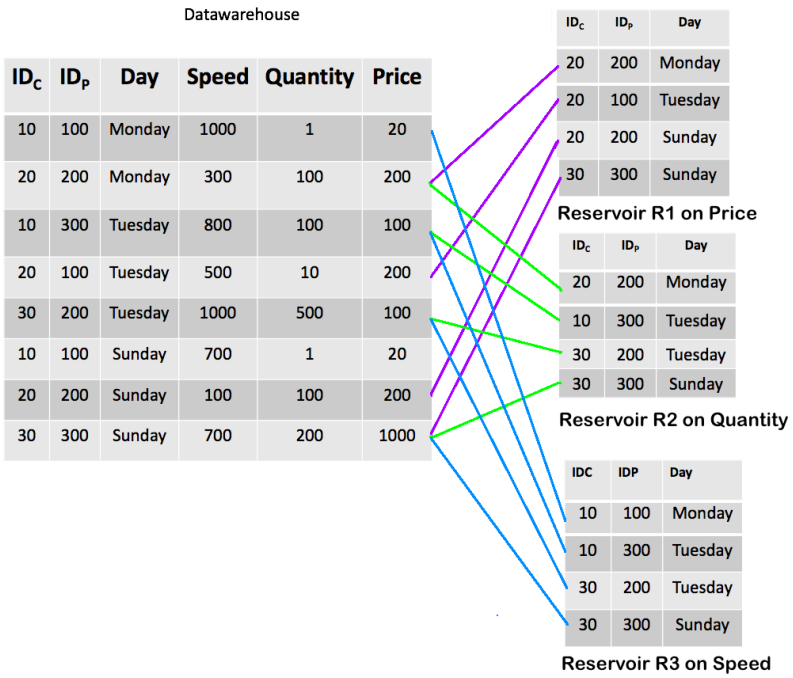

FIGURE 4.5: Échantillons provenant d'un entrepôt de données pour les trois *Mesures*.

4.3.1 Préférences comme distribution sur les *Mesures*

Soit Q_C^1 la distribution obtenue par l'analyse sur la dimension C pour la première *Mesure* M_1 et $\widehat{Q_C^1}$ l'approximation sur le Réservoir R_1. De la même façon, Q_C^2 est la distribution obtenue par l'analyse sur la dimension C pour la deuxième*Mesure* M_2 et $\widehat{Q_C^2}$ est l'approximation sur le Réservoir R_2.

Considérons une nouvelle *Mesure* M_α, définie à tout moment, et le Réservoir R_α défini dans la section 4.3. Soit $\widehat{Q}_C^\alpha$ la réponse de Q_C sur R_α et montrons le lien entre ces deux valeurs.

Theorem 1. *Soit Q_C^α la réponse de Q sur la dimension C pour la* Mesure *M_α. L'espérance de $\widehat{Q}_C^\alpha$ est Q_C^α.*

Démonstration. Evaluons $I\!E(\widehat{Q}^\alpha_{C=c})$, l'espérance de la densité pour $C = c$ sur le Réservoir R_α. C'est le nombre d'échantillons prévu avec $C = c$ divisé par k le nombre total d'échantillons. Le nombre attendu d'échantillons est :

$$\sum_t 1_{C=c}(t) \cdot Prob[t \in R_\alpha] = \sum_{t:t.C=c} (\alpha.\frac{k.t.M_1}{T} + (1-\alpha).\frac{k.t.M_2}{T})$$

car chaque t tel que $C = c$ dans R_α est pris avec probabilité $(\alpha.\frac{k.t.M_1}{T} + (1-\alpha).\frac{k.t.M_2}{T})$. Par conséquent :

$$\mathbb{E}(\widehat{Q}^{\alpha}_{C=c}) = \frac{\sum_{t:t.C=c}(\alpha.\frac{k.t.M_1}{T} + (1-\alpha).\frac{k.t.M_2}{T})}{k}$$

$$\mathbb{E}(\widehat{Q}^{\alpha}_{C=c}) = \frac{\sum_{t.C=c}(\alpha.t.M_1 + (1-\alpha)t.M_2)}{T}$$

$$= Q^{\alpha}_{C=c}$$

L'espérance de la densité $\widehat{Q}^{\alpha}_{C=c}$ est précisément $Q^{\alpha}_{C=c}$. □

Theorem 2. *Il existe (ε, δ) tel que Q^{α}_{C} sur la dimension C peut être (ε, δ)-approximé par $\widehat{Q}^{\alpha}_{C}$.*

On peut approximer $Q^{\alpha}_{C=c}$ en prenant plusieurs Réservoirs indépendants et appliquer Chernoff-Hoeffding. Avec l'inégalité de Markov nous obtenons une approximation plus faible, comme dans lemma 4.

4.3.2 Conclusion

Dans cette partie, nous avons montré comment approximer les requêtes OLAP, à partir d'un entrepôt de données en streaming, à l'aide d'un tirage proportionnel aux *Mesures*. Nous avons introduit une notion de préférence comme une distribution sur les *Mesures*. Nous avons également montré, que nous pouvons généraliser l'approximation sur toute préférence.

5 Analyse du réseau social Twitter en streaming

De nombreux réseaux sociaux partagent des flux de données avec des utilisateurs, par exemple Twitter, un réseau social de type microblogging créé en 2006. Les utilisateurs publient des messages appelés *tweets* qui sont partagés selon certaines règles simples. Il a été conçu à l'origine comme un service basé sur le SMS où les messages étaient limités à 160 caractères. Aujourd'hui, les *tweets* sont limités à 280 caractères, laissant 20 caractères pour le nom de l'utilisateur.

L'étude des Tweets est un sujet classique en particulier pour *l'analyse de sentiments*. On cherche à savoir pour un sujet donné si le Tweet est positif ou négatif. L'analyse de sentiments est présentée dans [38, 30].

Dans la première section, nous expliquons comment récupérer les données par des API. Dans la deuxième section nous décrivons le type de données qu'on peut récupérer. Dans la troisième section, nous décrivons comment le flux de tweet est transformé en un flux d'arêtes d'un graphe. Dans la quatrième section, nous montrons comment décider de manière approchée certaines propriétés de graphes Twitter.

5.1 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. Dans notre cas, nous utilisons l'API Streaming.

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

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

5.1.1 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" écrits 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.

La search API présente des limites :

1. Le nombre de tweets retournés par requête ne peut pas dépasser 1500 tweets.
2. Il est impossible de retrouver 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]

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.

```python
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 5.1: Exemple d'utilisation de l'API search en Python

5.1.2 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 : Il donne 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 que vous souhaitez.
3. Flux de site : Les flux de site sont utilisés par les serveurs qui se connectent à Twitter en tant que serveur proxy [4].

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

4. De nombreux sites web utilisent ce point d'accès pour afficher les tweets dans une page web.

Pour l'analyse en temps réel des données de Twitter, les flux publics constituent une bonne source d'API.

5.2 Caractéristiques des données Twitter

Toutes les API Twitter qui renvoient des tweets fournissent ces 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.

5.2.1 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 5.1: Exemple de Twitter JSON

5.2.2 Contenus

Sur Twitter, nous recevons de nombreux objets, les Tweets et leurs auteurs. Ces objets encapsulent tous les attributs principaux qui décrivent l'objet. Chaque Tweet comporte

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

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 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".

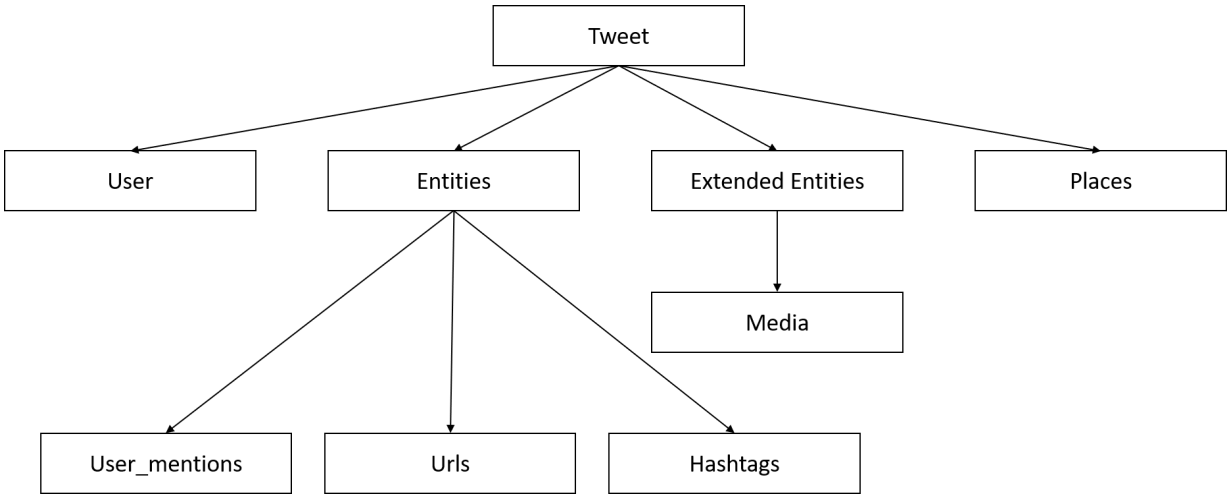

FIGURE 5.2: Exemple d'arborescence Twitter

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

Nous distinguons deux grands types de graphe possibles sur Twitter : le graphe classique d'utilisateur où les arêtes déterminent les relations entre les utilisateurs 5.4. Un autre graphe possible est déterminé par des événements. Nous définissons 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.

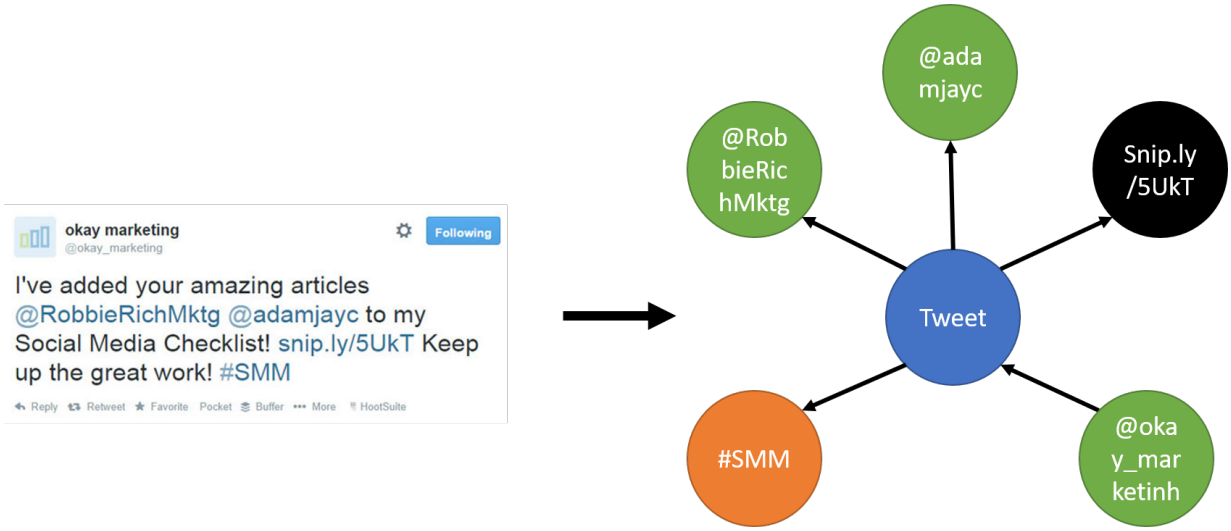

FIGURE 5.3: D'un tweet à un graphe

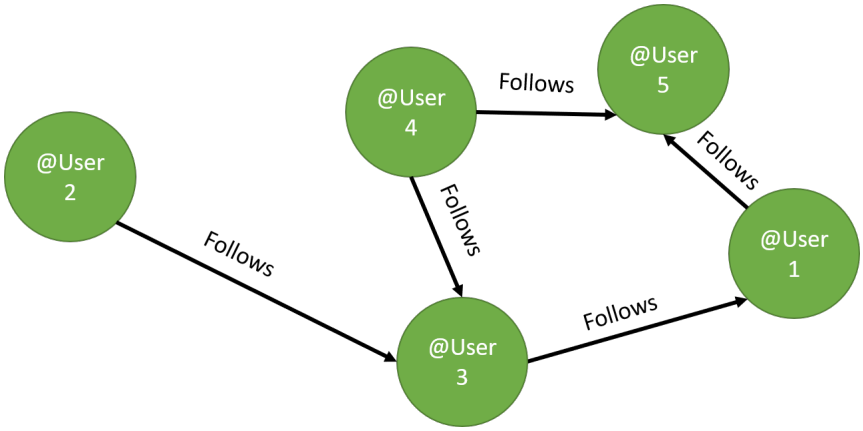

FIGURE 5.4: Exemple de graphe non symétrique d'utilisateurs sur Twitter

Dans la suite, nous considérons les graphes de tweets générés sur un sujet donné, défini par un ou plusieurs tags.

5.3.1 Twitter graphe d'un flux de données

Dans les réseaux sociaux et le crowdsourcing, nous observons d'importants flux de données en ligne, le plus souvent des arêtes $e_1, ..., e_m$ d'un graphe. Compte tenu d'un ensemble de tags tels que {#Ethereum, #Bitcoin}, ou {#Amazon}, Twitter fournit un flux de tweets représenté par des arbres JSON dont le contenu C (le texte du tweet) contient au moins un de ces tags. Le *Graphe Twitter* du flux est le graphe $G = (V, E)$ avec des arêtes multiples E où V est l'ensemble des tags $\#x$ ou $@y$. Pour chaque tweet envoyé par $@y$ qui contient des tags, nous ajoutons les arêtes $(@y, \#x)$ et $(@y, @z)$ dans E.

Dans notre approche, nous considérons l'hypergraphe où nous ajoutons le contenu C à chaque arête. Nous avons donc les hyperarêtes $(@y, \#x, C)$ et $(@y, @z, C)$. Les URL qui apparaissent dans le tweet peuvent également être considérées comme des nœuds mais nous les ignorons pour des raisons de simplicité. Un flux de tweets est ensuite transformé en un flux d'arêtes $e_1, ...e_m, ...$, bien que chaque arête soit une hyperarête, qui stocke également un timestamp.

Les réseaux sociaux tels que Twitter évoluent de manière dynamique, et des sous-graphes denses apparaissent et disparaissent au fil du temps à mesure que l'intérêt pour des événements particuliers augmente et disparaît.

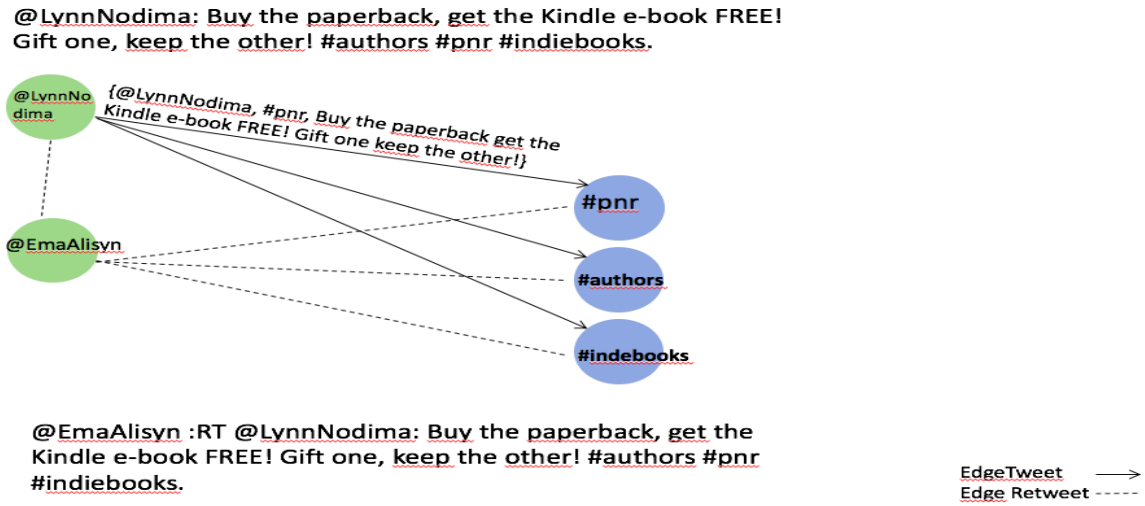

FIGURE 5.5: Graphe Twitter

5.4 Propriétés d'un graphe Twitter

L'analyse des réseaux sociaux utilise plusieurs concepts issus de la théorie des graphes qui fournit des outils pour représenter formellement les réseaux sociaux et quantifier leurs propriétés structurelles. Les graphes des réseaux sociaux satisfont certaines propriétés générales, comme une distribution particulière de degrés présentée dans la section 5.4.1 et un petit diamètre. On étudie dans la section 5.4.2 des solutions algorithmiques efficaces pour ces classes de graphes.

5.4.1 Distribution des degrés

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

$$P(k) = \frac{|\{u_i \in V : deg\, u_i = k\}|}{|V|}, \tag{5.1}$$

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

Dans les graphes aléatoires Erdös-Rényi, il a été montré dans [3] que $P(k)$ suit une distribution de Poisson dont le pic est situé à $\langle k \rangle$ ($\langle . \rangle$ désigne la valeur d'attente).

Une observation générale des réseaux sociaux est de constater que tous les graphes suivent une distribution de degrés comme le montre la figure 5.6. La figure 5.7, illustre la distribution en loi de puissance des degrés dans un graphe Twitter. Dans une loi de puissance, la probabilité pour un nœud donné d'avoir k arêtes est définie comme suit :

$$P(k) \sim k^{-\lambda} \tag{5.2}$$

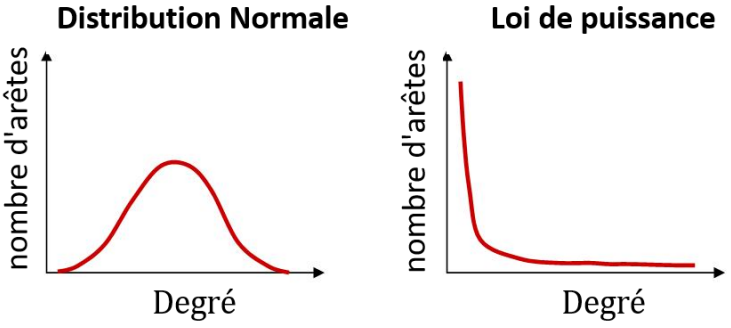

FIGURE 5.6: Distributions de degrés

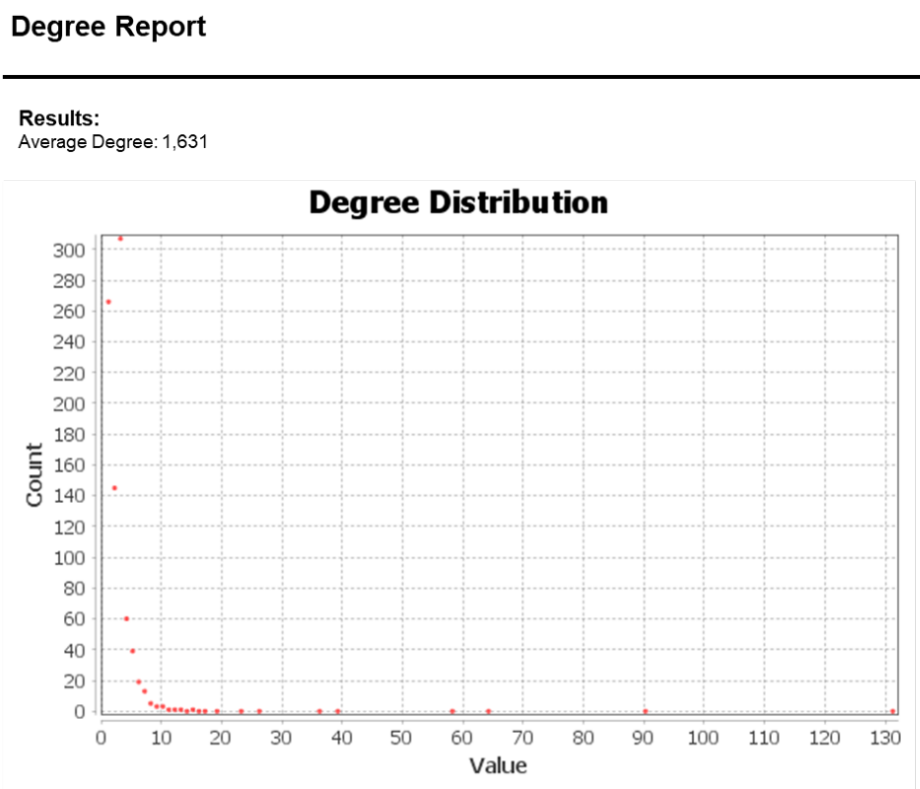

FIGURE 5.7: Exemple de la distribution de degrés dans un graphe Twitter

Notre modèle de graphe a une distribution de degrés qui suit une loi de puissance, telle que la loi de Zipf où :

$$Prob[d(i) = j] = \frac{c}{j^2} \; avec \; c = \frac{6}{\pi^2} \tag{5.3}$$

car :

$$\sum_{i \geq 1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$$

5.4.2 Propriétés à analyser

Pour un graphe social G, il existe de nombreuses propriétés qui peuvent ou ne peuvent pas être satisfaites. Par-exemple l'existence de grands sous-graphes denses, une variante du problème *Maxclique*.

Si on suit plusieurs flux, par exemple deux flux, on obtient deux graphes sociaux G_1 et G_2. Comment peut-on étudier leur corrélation ? Nous étudions cette question dans la section 5.4.2.3.

5.4.2.1 Communautés et sous-graphes denses

Soit $S \subseteq V$ et $E(S)$ l'ensemble des arêtes internes, c'est-à-dire les arêtes $e = (u, v)$ où $u, v \in S$.

Definition 5. *Un* γ-cluster S *est un* sous-graphe *qui dépend d'un paramètre* $\gamma \leq 1$ *tel que* $|E(S)| \geq \gamma.|S|.|S-1|/2$.

Un sous-ensemble S est (δ)-*grand*, si la taille de S est plus grande que $\delta\sqrt{n}$. Le problème (γ, δ)-*grand sous-graphe dense* consiste à décider s'il existe un γ-cluster S qui est (δ)-*grand*.
Si $\gamma = 1$, l'ensemble S est une clique. En général ce problème généralise *MaxClique*, un problème NP-dur, qui est difficile à approcher [31].

Definition 6. MaxClique *est un problème d'optimisation, qui consiste à trouver la taille de la plus grande clique d'un graphe.*

On peut aussi considérer *MaxClique* comme un problème de décision. L'entrée est un graphe G_n et un entier $\lambda \leq n$. On décide s'il existe une clique de taille plus grande que λ.

La densité classique de S est le rapport $\rho = |E[S]|/|S|$. On peut vouloir trouver des sous-graphes avec des nœuds S qui maximisent ρ. Dans le cas d'un flux d'arêtes, l'approximation des sous-graphes denses est bien étudiée dans [8, 26, 29, 43] et une limite inférieure de l'espace $\Omega(n)$ est connue [6].
Les deux versions du problème sont difficiles à approximer, et nous allons modifier la notion d'approximation pour la rendre compatible avec la classe de graphes sociaux, en utilisant la première version de graphes denses. Une famille de graphes G_n a un (γ, δ)-cluster s'il existe (γ, δ) tel que pour tout n assez grand $\exists S$ tel que S est un (γ, δ)-cluster pour G_n.
Soit μ la distribution uniforme des graphes qui suivent une loi de degrés comme une loi de puissance. Soit un langage L associé à un problème de décision, c'est-à-dire L est l'ensemble de toutes les instances positives.

Definition 1. *Un algorithme randomisé* A *est* asymétrique pour la distribution μ *pour décider le langage* L *s'il satisfait les deux conditions :*

- *Pour tout* $x \in L$, $Prob_{\Omega}[A(x)\ accepte] \geq 1 - \varepsilon$
- *Si* $x \notin L$ *est tiré selon* μ, $Prob_{\mu \times \Omega}[A(x)\ rejette] \geq 1 - \varepsilon$

L'approximation n'est plus pour le cas le plus défavorable. La deuxième condition spécifie une probabilité de rejet pour une entrée négative *tirée selon* μ.
L'algorithme A va considérer la taille des composantes géantes du Réservoir, pour une taille de k bien choisie et décider la propriété P : *Il existe un* (γ, δ)-*cluster ?*

Algorithme A1

Entrée : un flux de m arêtes d'un graphe G.
Sortie : Accepter s'il existe un (γ, δ)-cluster, sinon Rejeter. • Maintenir un Réservoir R de taille $k = c.\sqrt{n}\log n/(4 * \gamma\delta)$.
• Soit C la plus grande composante connexe de R.
• Accepter si $|C| \geq \Theta(n^{1/8}\log^2 n)$, sinon Rejeter.

Détection de communautés. Soit C la plus grande composante connexe d'un Réservoir de taille $k = (\alpha\sqrt{n}\log n)$. Considérons l'algorithme $A1$, qui dépend de γ, δ où $\alpha = 1/\gamma\delta$ est un paramètre auxiliaire.

La borne $n^{1/8}\log^2 n$ est une application directe du théorème 5.4.2.1 de Molloy-Reed. On va montrer que $A1$ est un algorithme asymétrique pour la distribution μ. La première condition suit le résultat suivant :

Théorème 1. *Si G a $O(n.\log n)$ arêtes et un (γ, δ)-cluster S, l'algorithme $A1$ Accepte presque surement.*

Démonstration. Pour un graphe G avec $cn.\log n/4$ arêtes, comme ceux dont la distribution de degrés est une loi de puissance, considérons le taux d'échantillonnage :

$$\frac{k}{m} = \frac{c.\sqrt{n}\log n/4}{\gamma.\delta.cn.\log n/4} = \frac{1}{\gamma.\delta.\sqrt{n}}$$

Celui-ci est indépendant des arêtes. On a donc dans le Réservoir un graphe tiré selon Erdos-Renyi avec une probabilité de $\frac{k}{m}$.

Puisque

$$|S| > \delta.\sqrt{n}$$

$$\frac{k}{m} = \frac{1}{\gamma.\delta.\sqrt{n}} > \frac{1}{\gamma.|S|}$$

$$\frac{k}{m} = \frac{1}{\gamma.\delta.\sqrt{n}} > \frac{(1+\epsilon)}{\gamma.|S|}$$

D'après le lemme 2, il existe une composante géante si

$$\frac{k}{m} > \frac{(1+\epsilon)}{\gamma.|S|}$$

L'algorithme $A1$ accepte donc presque sûrement. □

Non Détection. La deuxième condition à vérifier est plus difficile. L'étude est basée sur les graphes aléatoires satisfaisant une distribution de degrés, comme loi de puissance. Le résultat central de Molloy-Reed [48] donne une condition suffisante pour la non existence

de composantes géantes et plus précisément une borne sur la taille de la plus grande composante connexe.

Soit $d_i(n)$ le nombre de noeuds de degré i pour un graphe de taille n. La distribution $(d_i(n))_{i,n}$ est réalisable si :

1. pour tout n, il existe un graphe de taille n dont la distribution de degré est $(d_i(n))_i$,
2. pout tout i, $\ell_i = \lim_{n\to\infty} d_i(n)/n$ existe.

Considérons une distribution réalisable telle que :

1. $Q(\mathcal{D}) = \sum_i (i^2 - 2i)\ell_i$ est plus petit qu'une constante inférieure à 0,
2. le degré maximum est inférieur $n^{1/9}$,
3. le degré moyen est $O(1)$.
4. la convergence de $i(i-2)d_i(n)/n$ vers sa limite ℓ_i est uniforme.

Soit μ la distribution des graphes aléatoires qui suivent les $d_i(n)$ dans le modèle de configuration. [48] montre que presque surement, la taille de la plus grande composante connexe est au plus $Bn^{1/4}$ pour une constante B qui dépend de $Q(\mathcal{D})$. D'autre part aucune composante connexe de G a plus d'un cycle et il y a au plus $2Bn^{1/4}$ cycles.

Pour une instance aléatoire selon μ, la taille de la plus grande composante connexe est inférieure au seuil de l'algorithme $A1$. Notez que la variable n de Molloy-Reed s'applique au nombre de noeuds du graphe du Reservoir, environ $\sqrt{n}$. D'où le seuil $n^{1/8}\log^2 n$, utilisé par l'algorithme $A1$.

Théorème 2. *[47] Soit G un graphe aléatoire tiré uniformément selon la loi μ. l'algorithme $A1$ Rejette presque surement et est correct.*

On peut trouver des entrées adversaires dans ce cas sont des graphes, qui n'ont pas de (γ, δ)-cluster pour lesquelles l'algorithme se trompe dans le sens il va dire il existe un (γ, δ)-cluster avec une certaine probabilité. Ces entrées ne suivent pas la loi de puissance. L'approximation est donc dans ce sens précis.

Ce résultat se généralise aux graphes dynamiques générés par des fenêtres coulissantes. On détectera les composantes géantes et on pourra suivre leurs évolutions dans le temps.

5.4.2.2 Graphes dynamiques dans un flux d'arêtes

Nous suivons le modèle des fenêtres glissantes, comme le montre la figure 5.8, 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.

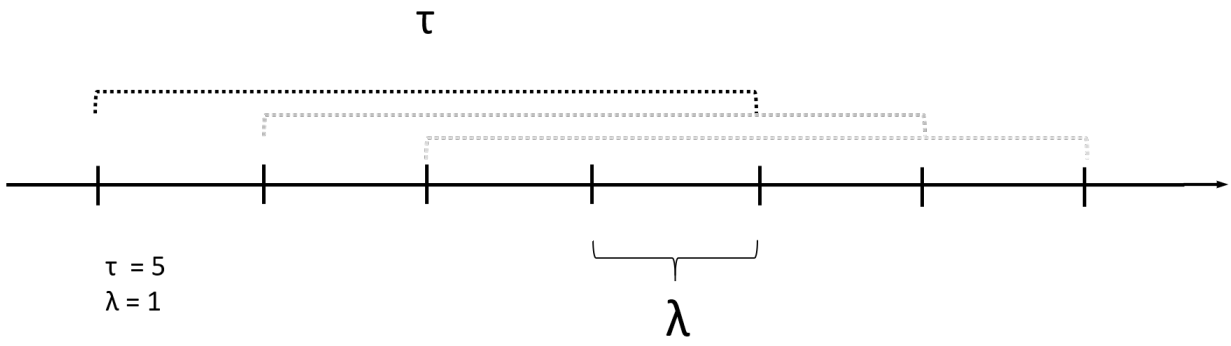

FIGURE 5.8: Step reservoir sampling.

5.4.2.3 Corrélation de contenu entre flux

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

$$\frac{I\!E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X . \sigma_Y} \tag{5.4}$$

Comment peut-on l'étendre aux graphes ? On pourrait choisir certains paramètres des graphes et prendre les corrélations entre ces paramètres. Les graphes sociaux ont cependant des statistiques très similaires et la corrélation serait importante. Le noyau de l'information semble se cacher dans la structure des clusters.

Soit deux graphes G_1 et G_2 issus de deux flux distincts. Une première approche de leur corrélation de contenu serait de considérer la similarité Jaccard [6] $J(V_1, V_2)$ sur les domaines des deux graphes. Cette définition a 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 appliquons la similarité de Jaccard uniquement aux clusters des graphes. Soit $C_i = \bigcup_j C_{i,j}$ l'ensemble des clusters du graphe G_i, pour $i = 1$ ou 2.

6. 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 $1 - J(A, B)$.

Definition 7. *La corrélation de deux graphes est le coefficient* $\rho = J(C_1, C_2)$.

Cette approche exploite la structure des graphes, est insensible au bruit et s'adapte bien lorsque les graphes ont des tailles différentes. Dans la suite, on considère pour simplifier que les graphes ont un seul cluster. Il est alors possible d'approximer la corrélation en utilisant les composantes géantes des graphes. Il suffit de considérer le 2-core de chaque composante géante. En effet, comme le montre [47], le 2-core de la composante géante du Réservoir est une bonne approximation, au sens ensembliste, de chaque C_i.

Algorithme A2

Entrée : deux flux d'arêtes définissant les graphes G_1, G_2.
Sortie : une approximation de la corrélation ρ.
• Maintenir deux Réservoirs R_1, R_2 de taille $k = \Theta(c.\alpha\sqrt{n}\log n/4)$.
• Soit $\widehat{C_1}$ la plus grande composante connexe de R_1 et $\widehat{C_2}$ la plus grande composante connexe de R_2.
• Soit $\widehat{C_1'}$=2-core($\widehat{C_1}$) et $\widehat{C_2'}$=2-core($\widehat{C_2}$).
• Renvoyer $J(\widehat{C_1'}, \widehat{C_2'})$

On peut donc approximer chaque C_i par $\widehat{C_i'}$ et ainsi la similarité de Jaccard.

Théorème 3. *L'algorithme A2 approxime le coefficient* $\rho = J(C_1, C_2)$.

Pour chaque fenêtre on fait cette analyse. Dans le cas des fenêtres dynamiques la fonction de corrélation dépendra du temps et donnera une approximation sur chaque fenêtre.

5.4.3 Conclusion

Nous avons montré comment analyser les graphes Twitter. On cherche en particulier à décider s'il existe de grands clusters. Notre contribution est une implémentation de la méthode présentée dans [42] dans le contexte collaboratif Python de Google. L'environnement Colab, permet d'implémenter les algorithmes écrit en Python, dans un contexte logiciel des serveurs de Google qui lisent directement les données de Twitter. Ce développement logiciel a servi de motivation pour imaginer des échantillonnages sur des distributions plus sophistiquées dans le cadre de l'analyse du texte.

6 Analyse de texte en streaming

Le but du chapitre est de présenter un modèle statistique construit à partir des données de texte en streaming, sans stocker l'ensemble du texte.

Dans le cas d'un texte classique, chaque phrase génère des arêtes où les nœuds sont les mots et les arêtes relient deux mots de la phrase, éventuellement deux mots consécutifs. Nous étudions les grandes composantes de ces graphes dans des fenêtres coulissantes en échantillonnant un nombre fixe d'arêtes avec deux distributions différentes, la distribution uniforme ou une distribution pondérée. Ces distributions définissent des sous-graphes aléatoires du graphe d'origine. L'analyse est basée sur l'étude des composantes géantes de ces sous-graphes aléatoires.

Pour un texte standard, un flux RSS d'articles d'actualité peut générer un flux important de phrases. Le graphe généré peut donc être très grand. Nous observons que pour l'échantillonnage uniforme des arêtes, les composantes géantes ne sont pas stables. Nous observons aussi que si nous échantillonnons les arêtes proportionnellement à la similarité des mots, donnée par *Word2vec* [44], les composantes géantes deviennent stables. Nous pouvons également classifier le texte en introduisant une distance naturelle entre les composantes géantes et utiliser l'algorithme classique *k-means*.

La première section décrit les algorithmes de streaming adapté à ce contexte. La deuxième section détaille l'approche du texte en streaming. La troisième section introduit la stabilité. La quatrième section détaille la classification. La cinquième section étudie la gestion des préférences dans ce cadre.

6.1 Algorithmes de streaming

Les algorithmes de streaming introduits dans le chapitre 2 lisent les données sous forme d'un flux de taille n et maintiennent une petite mémoire : si possible constante indépendante de n, ou $poly(\log)n$ ou au moins sous-linéaire $o(n)$ dans la taille du flux. Les algorithmes classiques peuvent considérer un flux de valeurs numériques $x_i \in \{1, 2, ...n\}$, de mots sur un alphabet Σ, ou d'arêtes $e_i = (v_j, v_k)$ d'un graphe $G = (V, E)$ où $v_j, v_k \in V$.

Une technique importante appelée *Reservoir sampling* introduit dans la section 2.1.1 conserve k éléments du flux avec une *distribution uniforme*. Dans un flux de longueur m, chaque élément a une probabilité k/m d'être dans le Réservoir. Dans le cas d'un graphe

où les arêtes on des poids, le Réservoir conserve k éléments du flux avec une *distribution non uniforme*. Chaque élément e_i de poids w_i d'un flux à une probabilité $k \cdot w_i / \sum_i w_i$ d'être dans le Réservoir. Une autre situation est de lire des arêtes et d'avoir un oracle qui va donner le poids de chaque arête. Dans notre cadre, à chaque fois nous lisons une arête, nous faisons appel à la fonction *Word2vec* pour calculer la similarité entre deux mots. Si k est sous-linéaire, par-exemple $O(\sqrt{m})$, on obtient un algorithme d'espace sous-linéaire.

6.2 Représentation du texte en streaming

Supposons que nous lisions un texte sous forme de flux : $w_1, w_2, ...w_n....$ où w_i est un mot d'un vocabulaire important (10^4 mots environ). Pour chaque phrase, nous utilisons une étape de prétraitement, la *lemmatisation*, qui élimine les mots très fréquents (articles, verbes modaux), reconnaît les Entités (par exemple "World Trade Center") et normalise chaque mot. Nous échantillonnons ensuite deux mots w_i, w_j de la même phrase avec le poids :

$$v(w_i) \cdot v(w_j)$$

où v est la fonction *Word2vec* qui transforme un mot en un vecteur de dimension 300. La fonction *Word2vec* peut-être donnée comme un oracle ou peut-être construite tout en lisant le flux du texte. Nous pouvons alors associer un Réservoir de taille k, en utilisant l'échantillonnage pondéré, et étudier les composantes géantes du graphe défini par le Réservoir.

Le produit $v(w_i).v(w_j)$ est aussi appelé la *Similarité* de (w_i, w_j).

6.3 Stabilité

Considérons deux expériences sur le même flux (ou document D) avec deux Réservoirs *indépendants*. Soit V_1 (resp. V_2) l'ensemble des noeuds des composantes géantes de C_1 (resp. C_2) comme dans la section 2.3.4. Pour un flux D et un Réservoir de taille k, soit la variable aléatoire $\rho(D, k)$ définie comme suit :

$$\rho(D, k) = \frac{|V_1 \cap V_2|}{|V_1|}$$

La variable aléatoire dépend de deux expériences et on peut aussi prendre son espérance $\mathbb{E}(\rho(D, k))$. Pour les flux Twitter, l'échantillonnage uniforme offre une bonne stabilité, comme le suggère l'analyse théorique des composantes géantes. Au contraire, pour les textes, l'échantillonnage uniforme est instable comme on le montre dans la partie expérimentale du chapitre 9.

6.4 Classification

Nous introduisons d'abord une distance naturelle entre des composantes géantes, qui peut être généralisée à des ensembles de composantes. Nous pouvons ensuite appliquer l'algorithme *k-means*[1]

L'analyse du texte classique cherche à reconnaitre le contenue d'un texte en particulier quand on a k possibilité. Un problème classique dans l'analyse du langage comme dans le chapitre 3 est de pouvoir détecter si le contenu du texte fait référence à une classe particulière.

6.4.1 Distances entre les composantes géantes

Considérons deux composantes géantes $C_1 = (V_1, E_1)$ à l'instant t_1 et $C_2 = (V_2, E_2)$ à l'instant $t_2 \geq t_1$ sous forme de graphes étiquetés. Il existe plusieurs distances possibles :

- *La distance de Jaccard* $\mathsf{dist}_J(C_1, C_2) = 1 - J(V_1, V_2)$ où $J(V_1, V_2)$[2] est la similarité de Jaccard entre les domaines V_1 et V_2.
- *La distance d'edition* prend en compte les arêtes. Une *édition* est la suppression ou l'insertion d'une arête avec des étiquettes et $\mathsf{dist}_E(C_1, C_2)$ est le nombre minimum d'éditions pour transformer C_1 en C_2, divisé par $|E_1| + |E_2|$.
- La distance d'édition amortie tient compte de la différence de temps $|t_2 - t_1|$:

$$\mathsf{dist}_A(C_1, C_2) = \mathsf{dist}_E(C_1, C_2) + \alpha \cdot |t_2 - t_1|$$

pour une constante α.

Chaque fenêtre coulissante peut avoir plusieurs composantes géantes de tailles différentes. Dans ce cas, nous écrivons C comme une distribution sur les composantes avec un poids proportionnel à leurs tailles. Par exemple, si C a deux composantes C_1 et C_2, nous écrivons :

$$C = \frac{|C_1|}{|C_1| + |C_2|} \cdot C_1 + \frac{|C_2|}{|C_1| + |C_2|} \cdot C_2.$$

On peut donc écrire :

1. Étant donné un ensemble de points $(x_1, x_2, \ldots, x_n)$, on cherche à partitionner les n points en k ensembles disjoints $S_1, S_2, \ldots, S_k$ en minimisant la distance entre les points à l'intérieur de chaque partition :

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

où μ_i est le barycentre des points dans S_i. On procède par itération à partir de μ_1_i pour $i = 1, ...k$ aléatoires $\in (x_1, x_2, \ldots, x_n)$. On étiquète tous les points qui sont les plus proches de μ_1_i par i. On calcule ensuite les nouveaux barycentres des points étiquetés par i, soit μ_2_i. On réitère cette opération jusqu'à ce que les barycentres ne changent plus.

2. La similarité de Jaccard ou l'index entre deux ensembles A et B est $J(A, B) = |A \cap B|/|A \cup B|$.

$$C = \lambda \cdot C_1 + (1 - \lambda) \cdot C_2$$

Nous pouvons alors généraliser les distances entre les composantes aux distances entre les distributions des composantes. Dans la section suivante, nous n'utilisons que la distance de Jaccard.

6.4.2 Classification en k-classes

Une séquence de fenêtres coulissantes génère une séquence de composantes géantes $C_1, ...C_n$ avec toutes les distances entre les paires. Supposons pour simplifier que nous gardons la composante géante la plus grande. Nous pouvons ensuite les regrouper en k classes, avec l'algorithme classique *k-means*. Chaque classe i a un représentant C_i. Pour une nouvelle composante géante C, il suffit de vérifier le Min_i $\mathsf{dist}(C_i, C)$ pour classifier C.

6.5 Phrases centrales et mécanismes d'attention

Chaque composante géante peut être analysée, en partant du nœud de degré maximal, le *centre*, et ses arêtes adjacentes, les *arêtes centrales*. Pour le 2-core de la figure 6.3, nous commençons par le nœud central 1 de degré maximum et ses arêtes centrales adjacentes e_1, e_4, e_5, e_8. Chaque arête est de la forme *(u,v, texte)* et nous pouvons analyser la répartition de l'attention vue dans la section 3.4 [3] [60] des mots u et v dans chaque phrase *texte*.

Par exemple l'arête e_8 est :
(skate, jump, "A boy is jumping on skateboard in the middle of a red bridge.")
et l'arête e_4 est :
(skate, sidewalk, "The boy skates down the sidewalk.")
L'analyse de l'attention de la première phrase pour les mots *skate* et *jump* est donnée dans la figure 6.1. Une composante géante fournit un *nœud central* et des *phrases centrales*. On peut analyser la composante géante de manière récursive, en prenant parmi les noeuds explorés celui qui a le plus grand degré et itérer la construction. À l'étape suivante, le nœud 6 serait exploré avec trois nouvelles arêtes.

6.6 Préférences

Soit $S_1 = \{v_1(w) : w \in T_1\}$ l'ensemble des vecteurs associés aux mots w dans le processus *Word2vec* associé à un texte T_1, et $S_2 = \{v_2(w) : w \in T_2\}$ l'ensemble similaire pour un texte T_2. Les vecteurs peuvent être très différents et reflètent les différents sujets,

3. Pour un mot u, l'*attention de la phrase* est la distribution sur les autres mots avec un poids proportionnel à sa similarité Word2vec.

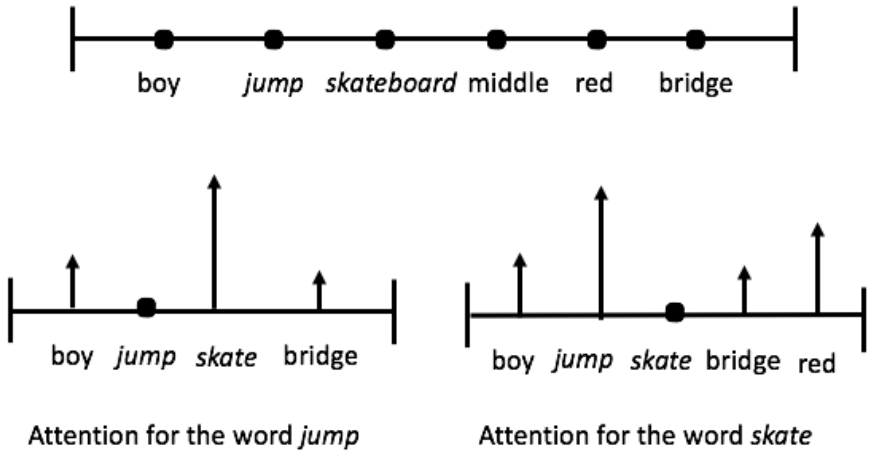

FIGURE 6.1: L'analyse de la phrase : *A boy is jumping on skateboard in the middle of a red bridge.*

comme le montre la figure 4.5. Une *préférence pure* est un tel ensemble de vecteurs construit à partir d'un texte de référence T. Une *préférence mixte* est une répartition sur S_i, par exemple $\lambda.S_1 + (1-\lambda).S_2$ pour 2 ensembles purs, i.e. l'ensemble des vecteurs :

$$S_\lambda = \{\lambda.v_1(w) + (1-\lambda).v_2(w) : w \in T_1 \cup T_2\}$$

si un mot $w \notin T$, nous supposons que $v(w) = 0$.

Nous montrons comment analyser un nouveau texte T comme un flux, en accordant une préférence mixte arbitraire. Comme dans l'exemple précédent d'un flux de données structurées, la distribution peut ne pas être connue à l'avance.

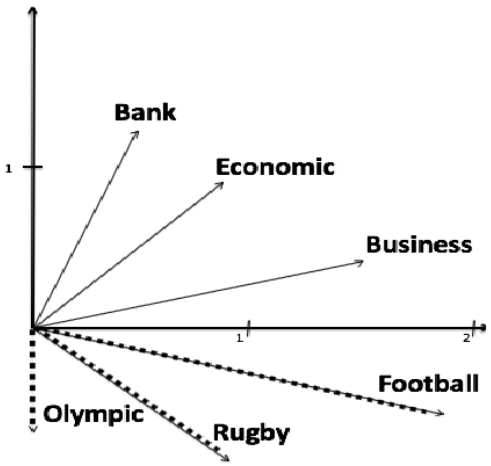

FIGURE 6.2: Deux ensembles de vecteurs dans 2 dimensions

Supposons que nous analysions un nouveau texte T comme un flux, en lui donnant une préférence mixte arbitraire, comme dans la figure 6.3. Nous maintenons deux Réservoirs R_1 pour les poids $v_1(w_i).v_1(w_j)$ et R_2 pour les poids $v_2(w_i).v_2(w_j)$. Nous pouvons construire un nouveau Réservoir R_α, comme dans la section 4.3.

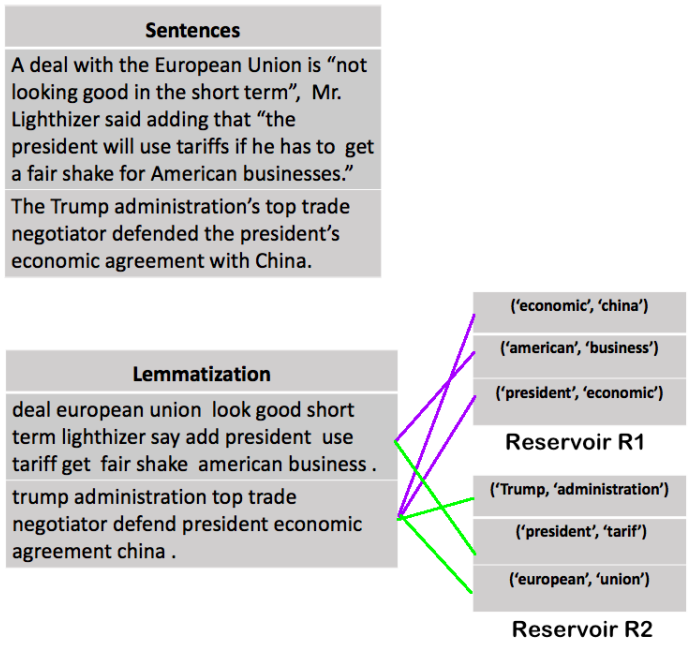

FIGURE 6.3: Deux Réservoirs pour un texte T

Theorem 3. *Le Réservoir R_α échantillonne des paires de mots (w_1, w_2) dans un flux, avec un poids proportionnel à $\alpha.(v_1, v_2)_1 + (1-\alpha).(v_1, v_2)_2$.*

Nous étudions les composantes géantes du graphe associées aux arêtes du Réservoir R_α. La figure 6.4 montre une composante géante typique et son 2-core. Le 2-core d'un Texte T est la structure centrale utilisée pour classifier le Texte, ou pour associer une thématique. Les distances entre les 2-core sont introduites dans la section 6.4.1.

6.7 Conclusion

Dans ce chapitre nous proposons une approche de l'analyse du texte en streaming, sans stocker tout le texte. Nous avons construit un graphe aléatoire dont les arêtes sont des échantillons de paires de mots d'une même phrase, choisis avec une distribution non uniforme. Chaque composante géante définit un sujet, qui peut être décrit par des mots centraux et des phrases centrales.

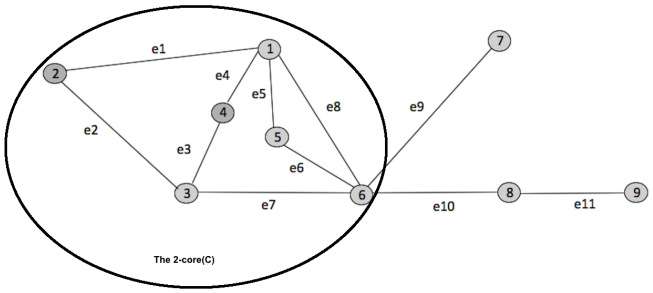

FIGURE 6.4: La composante géante du Réservoir R_α

7 Modèle de morphologie dans les langues naturelles

Les mots ont une structure interne, appelée aussi morphologie. Par exemple, le mot *préexistant* a un préfixe *pre-*, une racine *exist* et un suffixe *-ant*. Le suffixe *-ant* décrit l'information sur la catégorie syntaxique du mot. Dans ce cas, nous écrivons *pré-exist-ant* pour distinguer ces trois composantes. Considérons les verbes :

ap-porter, com-porter, col-porter, dé-porter, em-porter.

qui sont construits à partir de la racine *porter* avec différents préfixes qui changent le sens du mot. Si on associe un vecteur à chaque mot, quel est le rapport entre les vecteurs de deux mots qui ont la même racine mais deux préfixes différents par exemple *ap-porter, com-porter*?

Un sujet important est d'étendre les vecteurs des mots aux phrases et plus généralement aux paragraphes et aux textes. Nous construisons un vecteur *Word2vec* structuré, associé à chacun des mots mais qui va être composé de trois parties, une partie préfixe, une partie racine et une partie suffixe. Tous les vecteurs de l'exemple ci-dessus vont partager la même racine. Dans le chapitre précédent, on a considéré les vecteurs *Word2vec* comme des vecteurs donnés par des boites noires. On souhaite améliorer ces vecteurs, en les structurant et ce chapitre explique comment réaliser cette opération.

Dans notre modèle nous construisons des statistiques morphologiques qui vont permettre d'organiser nos vecteurs de manière structurée. Ce chapitre est basé sur l'article [16].

Des nouvelles approches essayent de généraliser les vecteurs des mots aux phrases et plus généralement aux paragraphes et aux textes. Dans notre cadre, nous étendrons des vecteurs aux phrases avec une approche complètement différente. Nous associons de nouveaux vecteurs à chaque phrase basés sur la structure des préfixes et suffixes qui capturent une information sémantique comme le Temps, la Voix, l'Humeur, la Force d'élocution. Cette approche, est particulièrement adaptée à certaines langues qui ont une forte morphologie comme l'*Amis*.

Dans la première section, nous présentons la langue *Amis*. Dans la deuxième section, nous présentons notre modèle statistique pour saisir la morphologie d'une langue naturelle et l'appliquer à l'*Amis*. Dans la troisième section, nous donnons un aperçu syntaxique

de *Amis*. Dans la quatrième section, nous étudions comment prédire approximativement la classe syntaxique d'une phrase simple. Dans la cinquième section, nous décrivons quel arbre de dérivation est préférable pour une grammaire donnée.

7.1 Langue avec une forte morphologie

Amis est l'une des quinze langues austronésiennes survivantes parlées à Taïwan. *Amis* est parlée le long de la côte orientale de Taïwan et compte quatre dialectes principaux présentant des différences significatives dans leur phonologie, leur lexique et leurs propriétés morphosyntaxiques. L'analyse porte sur les *Amis* du Nord : les données ont été recueillies par Isabelle Bril lors du travail de terrain [14] . Quelques phrases simples sont présentées en annexe A, où les affixes, les infixes et les clitiques [1] sont indiqués par des symboles spéciaux. Une propriété fondamentale de l'*Amis* est que les racines [2] sont le plus souvent sous-spécifiées et catégoriellement neutres [15] ; elles sont entièrement catégorisées (comme des noms, des verbes, des modificateurs, etc.) après avoir été dérivées et infléchies sous forme de mots morphosyntaxiques et projetées dans une phrase. Une autre propriété est que les préfixes, les suffixes, les affixes, contiennent énormément d'informations sémantiques et syntaxiques. Les bases nominales de noms sont signalées par l'article *u* ou par des démonstratifs. Les bases verbales ont des affixes de voix, les principaux étant la voix Acteur (Actor Voice) *mi-* (AV), la voix Patient (Undergoer Voice) *ma-* (UV), la voix Passive *-en*, la voix Locative *-an*. [19]. Un schéma syntaxique est donné dans la section 7.3.

L'Amis est une langue austronésienne, considérée comme une protolangue dans cette famille de 3000 langues parlées dans le Pacifique. De nombreuses études [14, 1] décrivent ces langues orales, leurs évolutions dans le temps et les grandes sous familles. Aujourd'hui, l'Amis est écrit en caractères chinois mais conserve sa structure linguistique unique. C'est un peu le *latin* des langues austronésiennes comme l'Indonésien et le Tagalog (langue parlée aux Philippines).

7.2 Un modèle statistique pour la morphologie

A partir de certains textes, nous pouvons analyser la distribution de fréquence des affixes. Avec une racine, la même analyse peut être effectuée. Nous appelons ces distributions statistiques les *statistiques morphologiques* de la langue. Il existe des approches similaires telles que [17] et [11] pour enrichir l'encastrement des mots avec des informations sur les caractères et les sous-mots. Dans notre cas, nous nous concentrons uniquement sur certains préfixes et suffixes. Nous considérons le premier et le deuxième moment des *statistiques morphologiques*. Nous pouvons alors déterminer quel préfixe est le plus probable dans un

1. le *clitique* est un affixe à l'intérieur d'un mot.
2. la *racine* est un mot atomique sans affixes. Les affixes sont soit inflexibles (c'est-à-dire qu'ils expriment une fonction sémantique ou syntaxique), soit dérivés (c'est-à-dire qu'ils créent différentes catégories).

mot manquant d'une phrase, quel suffixe est peu probable étant donné un préfixe et une phrase, et la structure syntaxique des phrases simples. Nous considérons que ces statistiques sont très utiles pour saisir approximativement certains paramètres sémantiques et syntaxiques clés. *Amis*, une langue naturelle où la morphologie est importante, se prête bien à cette analyse.

Nous avons construit un outil pour représenter la morphologie statistique de la langue *Amis*, étant donné un ensemble de textes où chaque mot a été décomposé en composants (c'est-à-dire préfixe, infixe, racine et suffixe).

- Nous analysons la distribution globale des racines et des affixes, c'est-à-dire les occurrences les plus fréquentes.
- Étant donnée une racine (ou un préfixe, ou un suffixe), nous obtenons la distribution des paires (préfixe,suffixe) de cette racine, et de manière générale nous analysons la distribution de paire préfixes, ou de paire de suffixes.

Compte tenu des nombreux textes oraux transcrits, nous avons d'abord construit un outil qui construit la distribution des préfixes, des suffixes et des racines, c'est-à-dire le nombre d'occurrences. A partir d'une racine, nous pouvons afficher la répartition de ses affixes. De même, nous pouvons donner un préfixe (respectivement un suffixe) et représenter la distribution des racines et des suffixes (resp. préfixes). Nous considérons alors les distributions de préfixes, suffixes et racines au second moment et construisons leurs représentations vectorielles. Notre modèle statistique combine ces trois représentations pour chaque mot composé de préfixes, d'une racine et d'un suffixe. Il introduit également un vecteur pour les phrases.

7.2.1 Statistiques de base pour la langue *Amis*

La répartition de tous les préfixes et suffixes, étant donné 70 textes d'*Amis* avec plus de 4000 mots, est donnée dans la figure 7.1. Tous les graphiques utilisent des valeurs absolues.

L'outils *Morphix* fournit une interface où une racine (resp. un préfixe ou un suffixe) peut être sélectionnée ; la distribution des préfixes et des suffixes pour une racine donnée est affichée graphiquement, comme dans la figure 7.2.

Étant donné la distribution des (préfixes ;suffixes) [3] de la figure 7.2, qui à ce stade ne sont pas ordonnées selon les formes et fonctions des racines, nous obtenons par projection la distribution des préfixes et suffixes de la figure 7.2 pour cette racine spécifique.

3. Un mot peut avoir plusieurs préfixes et suffixes. Dans la figure 7.2, les paires les plus fréquentes (préfixes ;suffixes) sont *(ma- ;)*, c'est-à-dire le préfixe *ma-* sans suffixe, *(ka- ;)*, c'est-à-dire le préfixe *ka-* sans suffixe, *(pa-se- ;)*, c'est-à-dire les deux préfixes *pa-* et *se-* sans suffixe, et *(ma ;ay)*, c'est-à-dire le préfixe *ma-* avec le suffixe *-ay*.

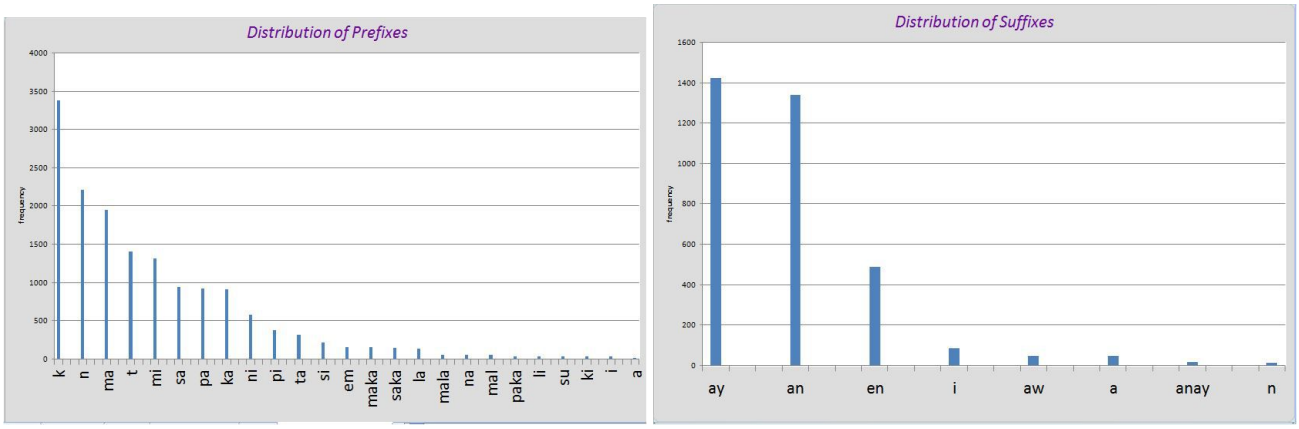

FIGURE 7.1: Préfixes et suffixes les plus fréquents.

7.2.2 Représentation vectorielle des préfixes, racines et suffixes

Étant donné une liste de préfixes n, la matrice de corrélation $M(n,n)$ mesure le nombre de cooccurrences des préfixes dans une même phrase : $M(i,j)$ est le nombre d'occurrences des préfixes i et j dans une même phrase et $M(i,i)$ est le nombre d'occurrences du préfixe i. Comme M est une matrice semi-définie positive, il existe une matrice $U(n,n)$ telle que $M = U.U^t$. Nous pouvons interpréter chaque préfixe i comme un vecteur v_i de dimension n tel que $v_i.v_j = M(v_i, v_j)$. Une méthode classique *PCA* (Principal Component Ananalysis) permet de réduire la dimension de ces vecteurs, le long de leurs composantes principales définies par les grandes valeurs propres telles que $v_i.v_j \simeq M(v_i, v_j)$. La SVD (Singular Value decomposition) produit les vecteurs propres (matrice S) et les valeurs propres λ_i (matrice diagonale V). Si l'on projette les vecteurs propres sur les dimensions définies par les grandes valeurs propres, nous réduisons la dimension.

Considérons les 4 phrases d'*Amis* suivantes, structurées avec la racine *padang* 'help, support' [4] :

1. *Mi-padang k-u tumuk t-u suwal n-ira tatakulaq.*
   AV-help NOM-ART chief OBL-ART word GEN-that frog [5]
   The tumuk supported the words of the frog.
2. *Isu Kungcu, yu ira k-u pa-padang-an,...*
   You Princess when exist NOM-ART RED-help-NMZ

4. La première ligne est le texte original où les préfixes et les suffixes sont identifiés. La deuxième ligne est l'analyse morphologique avec des étiquettes telles que AV, OBL,.... décrites ci-dessous. La troisième ligne est la traduction.

5. Abbreviations : AV Actor Voice ; ART article ; CV conveyance voice ; GEN genitive ; IMP imperative ; INST.V instrumental voice ; LOC locative ; LV locative voice ; NFIN non-finite ; NOM nominative ; NMZ nominaliser ; OBL oblique ; PFV perfect ; PROH prohibitive ; RED reduplication ; UV undergoer voice.

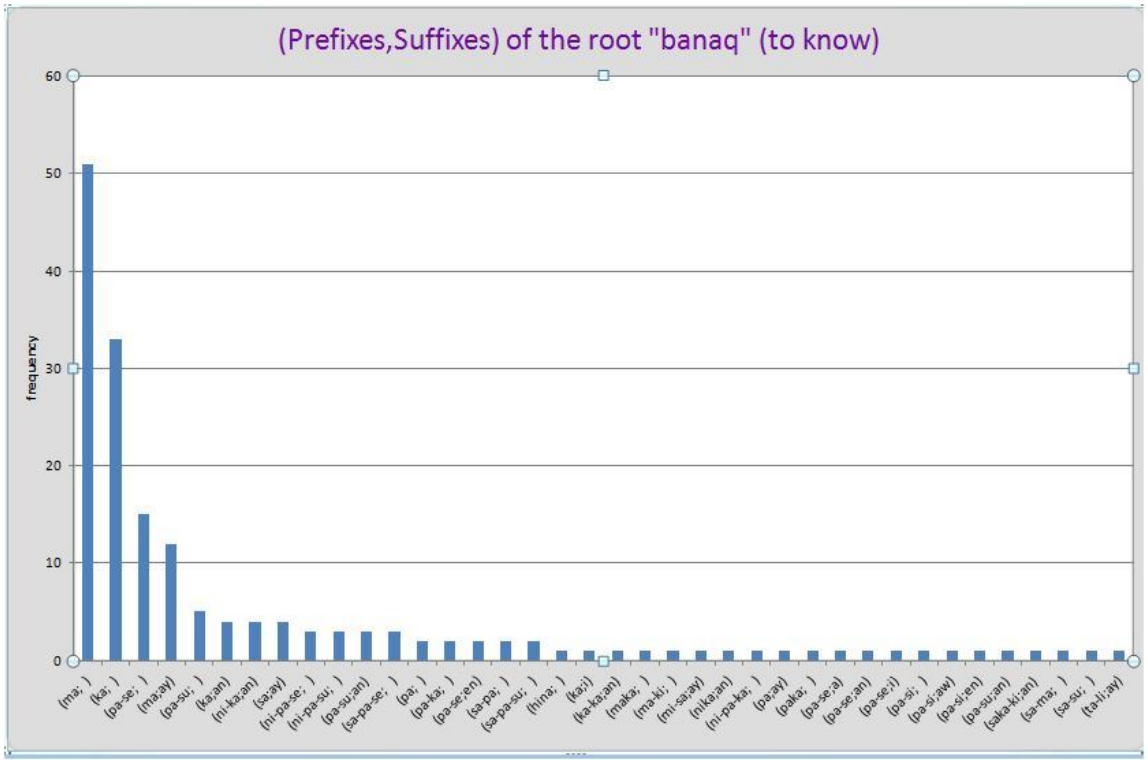

FIGURE 7.2: Les plus fréquents (préfixes ;suffixes) de la racine *banaq.* (' know').

You Princess, when (you) had some help,...

3. Sulinay mi-padang k-u taw,...

Indeed AV-help NOM-ART people

Indeed when people help,...

4. Aka-a ka-pawan t-u ni-padang-an n-u taw.

PROH-IMP NFIN-forget OBL-ART PFV.NMZ-help-NMZ GEN-ART people

Then, you mustn't forget people's help.

Dans ces quatre phrases, il y a sept préfixes dans l'ordre : *k,ka,n,ni,mi,pa,t* et des suffixes *-an, -a*. La matrice M_p pour ces préfixes est :

$$M_p = \begin{array}{c} \\ k \\ ka \\ n \\ ni \\ mi \\ pa \\ t \end{array} \begin{array}{c} \begin{array}{ccccccc} k & ka & n & ni & mi & pa & t \end{array} \\ \begin{pmatrix} 2 & 0 & 1 & 0 & 2 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 2 & 1 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 2 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 2 & 1 & 1 & 0 & 2 \end{pmatrix} \end{array}$$

La première ligne de la matrice indique 2 occurrences de *k-*, 1 occurrence de la paire *(k-, n-)* (première phrase), 2 occurrences de la paire *(k-, mi-)* (première et troisième phrases), 1 occurrence de la paire *(k-, pa-)* (deuxième phrase) et 1 occurrence de la paire *(k-, t-)* (première phrase). Les grandes valeurs propres de M_p sont de $2,5$ et $1,7$. Deux autres valeurs propres sont proches de 1 et les trois autres sont proches de 0. Si l'on décompose les vecteurs [6] sur les deux plus grands vecteurs propres, on obtient 7 vecteurs de dimension 2, un pour chaque préfixe dans la matrice ci-dessous et dans la figure 7.3.

$$B = \begin{array}{c} \\ k \\ ka \\ n \\ ni \\ mi \\ pa \\ t \end{array} \begin{array}{c} \begin{array}{cc} x_1 & x_2 \end{array} \\ \begin{pmatrix} -1.07140232 & 1.05796864] \\ -0.61507482 & -0.69205651 \\ -1.35624875 & -0.37685711 \\ -0.61507482 & -0.69205651 \\ -0.89917084 & 0.67091885 \\ -0.19923754 & 0.51352862 \\ -1.34624875 & -0.36685711 \end{pmatrix} \end{array}$$

$B * B^t$ est approximativement M_p. Le premier vecteur pour *k-* a les coordonnées $-1.07, 1.06$. Nous représentons graphiquement les 7 prefixes dans la figure 7.3. Une approche similaire peut être suivie pour les suffixes et pour les racines. La figure 7.3 peut être utilisée pour prédire, à partir d'un préfixe v, le préfixe suivant le plus probable v_{next}. C'est le vecteur v' qui maximise le produit scalaire $|v.v'|$. Par exemple, étant donné le vecteur pour le préfixe *mi-*, le préfixe suivant le plus probable est *k-*, dans ces 4 phrases. Pour les 95 phrases simples du corpus, les vecteurs seraient légèrement différents.

7.2.3 Distributions et vecteurs représentatifs

Soit δ la distribution des mots les plus fréquents, δ_P la distribution des préfixes les plus fréquents (resp. δ_R la distribution des racines) et soit π_p la projection qui associe le préfixe d'un mot. Par exemple, π_p(*mi-padang*)=*mi-*, un des principaux préfixes. De même, π_r associe la racine d'un mot, π_r(*mi-padang*)=*padang*. Ces distributions sont liées, principalement par des projections. Soit $\pi_p(\delta)$ la distribution telle que

$$\pi_p(\delta)(p) = \sum_{p \text{ est un préfixe de } w} \delta(w)$$

Puis $\delta_P = \pi_p(\delta)$ et $\delta_R = \pi_r(\delta)$. La matrice de corrélation M_p des préfixes est également la projection de la matrice de corrélation M des mots, c'est-à-dire $M_p = \pi_p(M)$.

Pour chaque matrice de corrélation M_p, M_r, M_s des préfixes, racines et suffixes, on applique la réduction de dimension et on obtient des vecteurs v_p de dimension n_p des préfixes, v_r de dimension n_r pour les racines et v_s de dimension n_s pour les suffixes. La réduction importante concerne M_r, dont la dimension est aussi grande que la taille du vocabulaire. Pour M_p, M_s, la dimension est petite et la réduction est moins importante. Dans l'exemple précédent, la dimension

6. On utilise un package d'algèbre linéaire en Python pour obtenir la décomposition SVD et les projections.

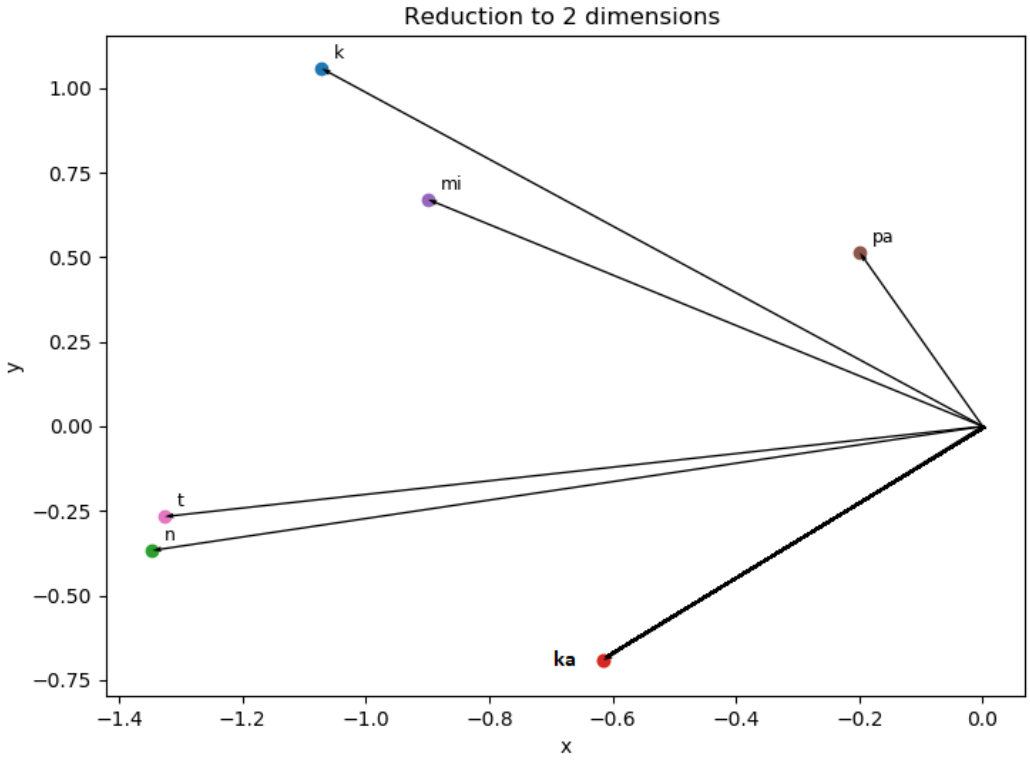

FIGURE 7.3: Les vecteurs des 7 préfixes les plus fréquents *k-,ka-,n-,ni-,mi-,pa-,t-* en deux dimensions.

n_p des préfixes est 7 sans réduction de dimension et de 2 avec la réduction. Considérons un mot *w=pre-root-suf* avec un préfixe et un suffixe. Soit $v_p(pre)$ le vecteur du préfixe *pre*, $v_r(root)$ le vecteur de la racine *root* et $v_s(suf)$ le vecteur du suffixe *suf*. Le vecteur $v(w)$ est l'union des trois vecteurs :

$$v(w) = \begin{pmatrix} v_p(pre) \\ v_r(root) \\ v_s(suf) \end{pmatrix}$$

Si on souhaite représenter des mots à plusieurs préfixes, on énumère les préfixes dans le vecteur $v(w)$. Pour deux mots w_i, w_j avec un préfixe, on définit :

$$\widetilde{M}(w_i, w_j) = M_p(pre_i, pre_j) + M_r(root_i, root_j) + M_p(suf_i, suf_j)$$

soit la somme des corrélations des préfixes, racines et suffixes. Le point fondamental est que pour deux mots quelconques w_i, w_j, le produit $v(w_i).v(w_j) \simeq \widetilde{M}(w_i, w_j)$. En effet, $v(w_i).v(w_j) = v_p(pre_i).v_p(pre_j) + v_{r,root_i}.v_{r,root_j} + v_{s,suf_i}.v_{s,suf_j}$. Le produit $v_p(pre_i).v_p(pre_j)$ correspond approxi-

mativement à $M_p(pre_i, pre_j)$ et de même pour les racines et les suffixes. D'où $v(w_i).v(w_j) \simeq \widetilde{M}(w_i, w_j)$. Pour les mots comportant plusieurs préfixes, cette approximation peut être généralisée.

Notez que $\widetilde{M}(w_i, w_j)$ peut être très différent de $M(w_i, w_j)$. Il est possible que $M(w_i, w_j) = 0$, mais que ses préfixes, suffixes et racines aient de fortes corrélations, d'où $\widetilde{M}(w_i, w_j)$ peut être grand. Une théorie riche de ces vecteurs structurés peut être élaborée à l'aide de corrélations croisées, que nous n'utilisons pas pour l'instant.

7.2.4 Vecteurs de phrases

Un sujet important est d'étendre les vecteurs des mots aux phrases et plus généralement aux paragraphes et aux textes. Les réseaux de neurones récurrents [32, 65] ou d'autres approches basées sur l'apprentissage [59, 58] construisent un tel vecteur de variables latentes. Nous suivons une approche différente car nous voulons capturer des propriétés plus générales des phrases à l'aide des préfixes et suffixes. Définissons le vecteur probabiliste s d'une phrase comme un vecteur de dimension 5 dont les composantes sont des distributions sur certains domaines finis :

- Valence : $\{0, 1, 2, 3\}$,
- Voix : {AV, UV, LV, INST.V},
- Temps : {Present, Past, Future},
- Mood : {Indicative, Imperative, Hortative, Subjunctive},
- Force d'élocution : {Declarative, Negative, Exclamative},

La première composante s^1, la *Valence*, est le nombre d'arguments du verbe. Nous la considérons comme une distribution sur le domaine $0, 1, 2, 3$ qui a 4 valeurs comme support. De même pour les autres composantes. Le vecteur s de dimension 5 a une taille globale de $18 = 4+4+3+4+3$. D'autres dimensions pourraient être utilisées, mais nous gardons ce type de vecteur pour simplifier. Si la troisième composante, une distribution sur le support {Present, Past, Future} est le vecteur $[0, 1, 0]$, elle indique un Past (avec une probabilité de 1). Si la composante était $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$, elle indiquerait une répartition uniforme sur les trois temps.

La figure 7.4 montre un vecteur sur la phrase :

5. mi-padang t-u suwal n-ira tatakulaq
'(he) supports the words of the frog.'

Nous lisons la phrase $w_1, w_2, ...w_n$, et le vecteur $v_i = v(w_i)$ associé avec chaque mot w_i, défini dans la section précédente. Soit :

$$s_i = F(s_{i-1}, v_i)$$

avec s_0 un état initial (vecteur zéro) et F une fonction. Nous construisons F par cas suivant les préfixes et les suffixes. Nous décrivons des règles syntaxiques plus avancées de l'*Amis* dans la section 7.3.

7.3 Un aperçu syntaxique de la langue *Amis*

L'ordre de base de l'*Amis* commence par le verbe. Les arguments sont marqués par les cas : nominatif (*k*-), génitif (*n*-), oblique (*t*-) [19]. Les affixes de voix (AV) *mi*-, (NAV) *ma*-, (UV) *ma*-,

Valence : {0,1,2,3}	[0,0,1,0]
Voix : {AV,UV,LV,INST.V}	[1,0,0,0]
Temps : {Present,Past,Fut}	[.9,.05,.05]
Mood : {Ind,Imp,Hort,Sub}	[1,0,0]
Force d'élocution : {Decl,Neg,Exclam}	[1,0,0]

FIGURE 7.4: Vecteur d'une phrase

identifient également des classes de verbes ; l'appartenance à une classe peut être exclusive ou permettre des alternances de voix avec une sémantique différente. Les verbes AV *mi-* désignent des activités. Les verbes UV *ma-* désignent des réalisations accomplies par un agent sur des patients spécifiques et totalement affectés ; les verbes NAV *ma-* comprennent des états, des propriétés. Le système de voix est basé sur la co-sélection d'un argument nominatif (le sujet), et d'un affixe de voix dont la sémantique correspond à celle du sujet nominatif. Les voix AV *mi-*, NAV *ma-* et UV *ma-* sont limitées à des phrases déclaratives. Les phrases non déclaratives (telles que les phrases négatives, impératives, hortatives) ont différentes formes. Comparons *ma-butiq cira* '(s)he is asleep/sleeping' et *ka-butiq !* 'go to sleep !'.

Transitivité et alignement [7]**.** Les verbes AV *mi-* et les verbes NAV *ma-* (Non-Actor Voice) ont un alignement de type "accusatif" avec un argument marqué par *t-* comme dans (6) et (8). Le sujet des verbes *mi-* est un acteur, tandis que le sujet des verbes "NAV" *ma-* est un non-acteur (c'est-à-dire un thème ou une expérience, le siège d'une propriété ou d'un état). D'autre part, les verbes UV*ma-* transitifs ont un alignement ergatif avec un sujet patient et un agent génitif comme dans (7). L'ordre des mots (V-S-0 ou V-S-A) est donné pour chaque exemple :

6. *Mi-melaw k-u wawa t-u tilibi.*
   AV-look NOM-ART child OBL-ART TV V-S-O
   ' The child is watching TV. '
7. *Ma-melaw n-uhni k-u teker.*
   UV-look GEN-3pl NOM-ART trap V-A-S
   'They saw the trap.'(lit. the trap was seen by him)
8. *Ma-hemek k-aku t-u babainay. (*mi-)*
   NAV-admire NOM-1sg OBL-ART boy V-S-O

7. L'alignement fait référence au type de marquage des arguments du verbe. Dans les langues accusatives, le sujet nominatif est marqué différemment de l'objet accusatif. Dans les langues ergatives, l'agent des verbes transitifs est marqué comme ergatif, tandis que le sujet des verbes intransitifs et le patient des verbes transitifs sont marqués de la même façon comme nominatif/absolutif.

'I admire the boy.'
Toutes les autres voix s'alignent de manière ergative, comme UV *ma-*, donc UV *-en*, LV *-an*, c'est-à-dire qu'elles ont un sujet nominatif correspondant à la sémantique de l'affixe de voix (c'est-à-dire un patient, un lieu), et un agent génitif (si exprimé).

7.4 Classification des phrases à partir des préfixes et suffixes

Les préfixes et suffixes apparaissant dans une phrase fournissent certaines informations syntaxiques. Ici, nous limitons l'analyse à 95 phrases simples, principalement sous forme de verbes AV et UV, données en annexe A. Nous codons chaque phrase comme dans la figure 7.5 ci-dessous, avec ses préfixes, les racines et les suffixes. Nous demandons ensuite s'il est possible de récupérer la syntaxe d'une phrase en observant uniquement les préfixes et les suffixes. Les racines lexicales des mots sont remplacées par un "—" comme dans la figure 7.5.

Sentences	**Sentences Encoding**
Ma-pa-tangasa n-umisu k-aku iri.	Ma-pa— n— k— —.
Adihay k-u ni-urung t-u kidudung.	— k— ni— t— —.

FIGURE 7.5: Les phrases et leurs encodages.

Les *simples clauses* considérées appartiennent à l'un des modèles syntaxiques suivants, d'une valeur de 10 :
On considère 10 types de *clauses simples* :

. *V-A-S-O.* Verb-Agent-Subject-Object
. *V-A-S.* Verb-Agent-Subject
. *V-A-O.* Verb-Agent-Object
. *V-A.* ...
. *V-S-O.*
. *V-S-A.*
. *V-O-S.*
. *V-S.*
. *V-O.*
. *V.*

Toutes les 95 phrases simples ont été étiquetées comme appartenant à l'une des 10 classes. Nous construisons maintenant un arbre de décision simple qui lit le codage d'une phrase et prédit approximativement sa classe. L'arbre de décision présentés dans la figure 7.7 décide d'abord si la phrase contient un préfixe *ma–* ou un préfixe *mi–* ou un préfixe *k–* sans préfixe *ma–* ou *mi–*. L'arbre détecte ensuite la présence ou l'absence des préfixes de cas *n–*, *k–* et *t–*. Une branche avec l'étiquette *n–* suppose la présence du préfixe *n–* et avec l'étiquette $\neg n$– suppose l'absence du préfixe *n–*. Chaque feuille de l'arbre est étiquetée avec l'une des classes syntaxiques.

Les statistiques qui représentent la fréquence de ces 10 classes différentes parmi le corpus de 95 phrases, sont données dans la figure 7.6.

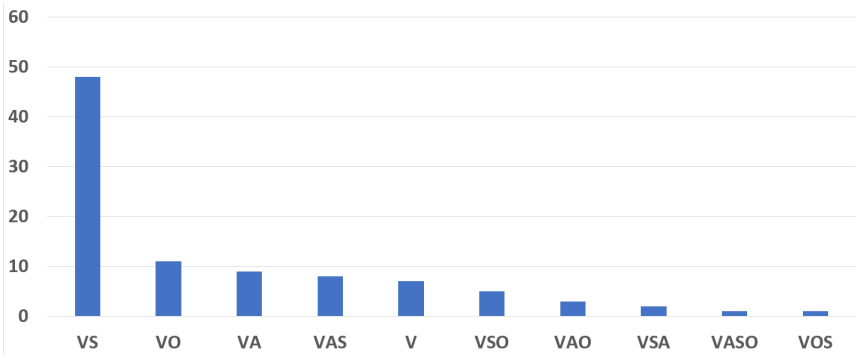

FIGURE 7.6: Fréquences des différentes classes syntaxiques existantes dans le corpus.

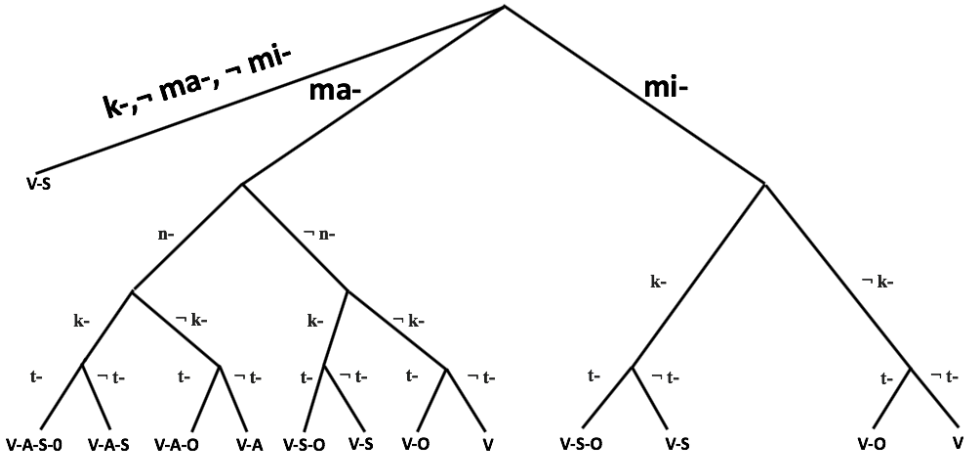

FIGURE 7.7: Arbre de décision pour prédire la classe syntaxique.

Dans les phrases de l'annexe A, il y a 95 phrases affirmatives simples et 4 phrases négatives. Parmi les 95 phrases affirmatives, 22 phrases n'affichent pas les préfixes *ma–* ou *mi–* ou *k–*, c'est pourquoi 73 phrases ont été classées par l'arbre de décision. La prédiction est considérée comme *moyenne* si la classe prédite est à une distance de 1 pour la distance d'édition avec la classe exacte, *mauvaise* si la classe prédite est à une distance supérieure à 1 de la classe exacte. Par exemple, *V-S-O* est une prédiction *moyenne* de la classe *V-A-S-O* car elle se trouve à une distance de 1. *V-S* est une mauvaise prédiction de *V-A-S-O* car elle se trouve à une distance de 2. Nous avons obtenu une bonne prédiction dans 25 cas, une prédiction moyenne dans 42 cas et une mauvaise prédiction dans 6 cas. Les statistiques de prédiction sur les 95 phrases sont données dans la figure 7.8.

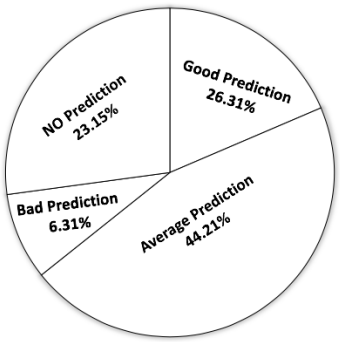

FIGURE 7.8: Statistiques des prédictions.

Ce simple arbre de décision classe 77% des 95 phrases présentées en annexe A. La plupart du temps, la classification est de qualité moyenne : elle ne donne qu'une approximation de la classe.

7.4.1 Vecteur de phrases et mécanismes d'attention

La valence, première composante du *vecteur de la phrase*, est définie pour chaque verbe (voir section 4). Les 4 autres composantes sont partiellement définies par les préfixes et suffixes et sont donc des distributions probabilistes. Comme le nombre total de préfixes et de suffixes est faible, la description de la fonction F est essentiellement une définition par cas.

Les mécanismes d'attention [40, 60] présentés dans le chapitre 3.4 prévoient pour un mot donné w_i d'une phrase, la distribution des autres mots les plus corrélés. La corrélation d'un mot w_j avec w_i est approximativement la valeur $v(w_i).v(w_j)$. On peut alors calculer $\{v(w_i).v(w_j) : j \neq i\}$ pour un w_i fixe et normaliser les valeurs pour obtenir une distribution. Le mécanisme d'attention est surtout utilisé pour la traduction automatique.

Nous pouvons nous concentrer sur les préfixes et les suffixes et définir de la même façon l'attention des préfixes et des suffixes comme le produit scalaire des vecteurs associés aux préfixes ou aux suffixes. Dans l'exemple de la figure 7.9, considérons la phrase : " *Mi-melaw k-u wawa t-u tilibi,*" avec les préfixes *mi–*, *k–*, *t–*. Calculons l'attention pour le préfixe *mi–*. Nous calculons $v_p(mi).v_p(k) = 1,149$ et $v_p(mi).v_p(t) = 1,446$, en utilisant les vecteurs donnés dans la matrice B de la section 7.2.2. Avec une normalisation, on obtient la distribution de la figure 7.9.

Pour les préfixes importants tels que *mi–* et *ma–*, la répartition de l'attention est importante et codifie certaines propriétés intentionnelles d'une phrase.

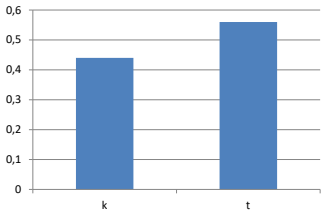

FIGURE 7.9: Distribution de l'attention pour *mi–* dans la phrase *Mi-melaw k-u wawa t-u tilibi.*

7.5 Grammaires et statistiques

Nous étudions maintenant l'impact des préfixes et suffixes d'une phrase sur les arbres de dérivation possibles, définis par les analyseurs syntaxiques des grammaires classiques. Étant donné une grammaire et une phrase, quels sont les arbres de dérivation les plus probables ? Il s'agit d'une question centrale pour tout analyseur, y compris les analyseurs de dépendance [18] construits par apprentissage à partir de données étiquetées. Dans ce cas, la distribution des exemples définit un espace probabiliste.

Un problème fondamental est d'associer à une séquence de symboles un arbre de dérivation possible, ce problème existe dans plusieurs disciplines : en biologie le repliement de structure ARN (Acide RiboNucléique) [8], en informatique l'arbre de décomposition d'une phrase. Un repliement de l'ARN est un arbre de dérivation possible avec une énergie potentielle et le repliement le plus probable a un niveau d'énergie minimum. Dans notre contexte, nous disposons de diverses statistiques sur les préfixes et les suffixes, et l'entropie de chaque distribution, obtenues à partir des données linguistiques. Nous n'avons pas de définition simple de l'énergie associée à un arbre et nous nous tournons vers l'interprétation linguistique, donnée en premier lieu dans la section 7.3.

Une grammaire G pour les mots représentés par la structure *préfixe-racine-suffixe* peut être représentée par des règles du type

$$\text{Non} - \text{terminal} \ \rightarrow \ \text{regular expression}$$

Un exemple typique d'utilisation de Non-terminaux [9] qui comprennent des mots, des préfixes, des racines et des suffixes est :

$$S \rightarrow VP.KP + VP.KP^*$$

8. Un repliement transforme une séquence linéaire de lettre en une structure avec des liens, un arbre qui est lui même ensuite transformé en une structure tridimensionnelle.

9. KP stands for Case Phrase, DP stands for Determiner Phrase, K stands for Case, PossP stands for Possessive Phrase. K and Voice represent Case prefixes and Voice prefixes respectively.

$VP \rightarrow Voice.V.KP^*$
$KP \rightarrow K.DP$
$DP \rightarrow D.N + D.N.PossP$
$PossP \rightarrow K.DP$

Un autre ensemble de règles énumère les mots et les affixes possibles.

$K \rightarrow t\text{–} \ +.....$
$V \rightarrow padang \ + ...$
$Voice \rightarrow mi\text{–} +.....$
$N \rightarrow suwal + ...$
$D \rightarrow u + ...$
$PossP \rightarrow n\text{-}ira +.....$

Chaque symbole non terminal $S, VP, KP, ...$ génère une expression régulière simple pour la première partie. Certains symboles non terminaux $K, V, ...$ énumèrent tous les mots possibles pour la deuxième partie. On peut générer une phrase appliquant plusieurs règles à partir de la racine S. L'arbre de dérivation est la construction inverse. Il existe plusieurs arbres de dérivation possibles associés à une phrase, telle que

mi-padang t-u suwal n-ira tatakulaq,

'(he) supports the words of the frog'

décrits dans la figure 7.10, et nous discutons de la meilleure représentation.

7.5.1 Les meilleurs arbres de dérivation

Étant donné un *vecteur de phrase* s, nous pouvons alors décider que l'arbre de dérivation (a) de la figure 7.10 est mieux adapté que l'arbre (b) pour la phrase présentée dans la section précédente. Nous reprenons en partie l'explication des verbes de la section 7.3.

Nous savons qu'un verbe dont le préfixe est *mi-* assigne un nominatif à l'Acteur 'voix' AV. La deuxième composante du vecteur s est : $[1, 0, 0, 0]$. Par conséquent, la structure VP de l'arbre de dérivation (a) encode tous les arguments alors qu'ils sont séparés dans l'arbre de dérivation (b). Si on a un Acteur 'voix', il faut dériver le verbe avec son KP. Si on a le préfixe *mi-*, l'action doit être attachée au verbe. Par conséquent, l'arbre de dérivation (a) est une meilleure représentation.

7.5.2 Comparaison avec d'autres modèles stochastiques

Dans une grammaire stochastique [41], les dérivations avec le même symbole non-terminal ont une probabilité p telle que la somme des probabilités pour chaque symbole non-terminal est de 1. L'espace probabiliste associé à l'ensemble (s,t), où s est une phrase et t est un arbre de dérivation, est le produit des probabilités des règles utilisées selon l'arbre, noté $p(s,t)$. Étant donnée une phrase, une tâche classique consiste à prédire l'arbre de dérivation le plus probable, et elle peut être réalisée en $O(n^3)$ pour une phrase de n mots .

Dans notre contexte, l'espace probabiliste est entièrement différent. Les vecteurs structurés des mots nous permettent de prédire le mot, préfixe ou suffixe le plus probable, compte tenu du contexte des mots précédents. Pour une phrase donnée, les positions des préfixes et suffixes

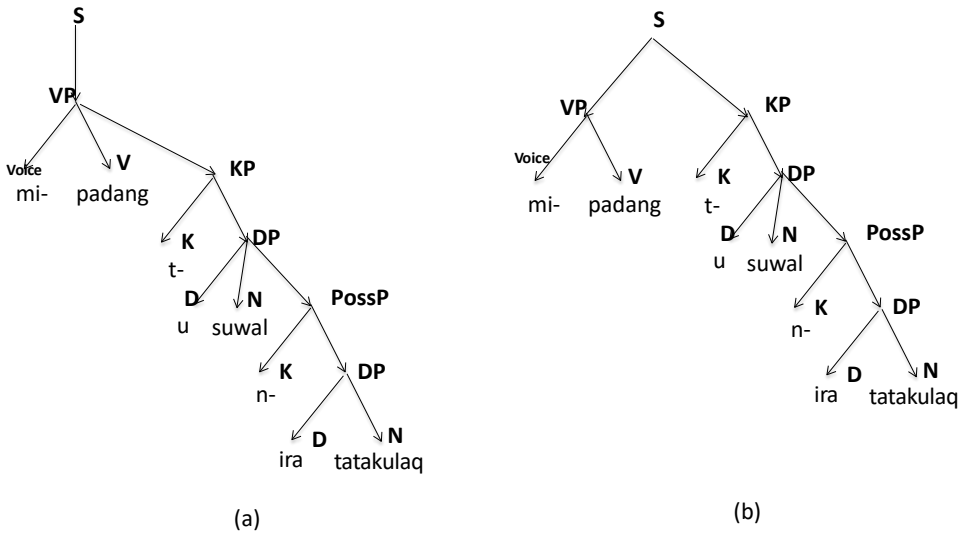

FIGURE 7.10: Arbre de dérivation de la phrase *mi-padang t-u suwal n-ira tatakulaq* pour une grammaire G.

déterminent les distributions probabilistes du vecteur s introduite dans la section 7.2.4. Nous examinons l'arbre de dérivation préférable, compte tenu de cette distribution des composantes sémantiques.

7.5.3 Conclusion

Nous proposons un modèle statistique pour la morphologie, inspiré par la langue *Amis*. La morphologie d'un mot w est la structure *préfixe-racine-suffixe* . Nous avons construit les distributions classiques des préfixes, racines et suffixes les plus fréquents, et les distributions correspondantes avec une racine, un préfixe ou un suffixe possible. On construit des vecteurs : $v_p(pre)$ pour le préfixe pre, $v_r(root)$ pour la racine $root$ et $v_s(suf)$ pour le suffixe suf. Le vecteur du mot $v(w)$ est la concaténation de ces trois vecteurs. Nous avons défini un vecteur probabiliste associé à une phrase. L'analyse des préfixes et suffixes, détermine la plupart des composantes du vecteur de la phrase S, de manière probabiliste. Nous avons montré comment prédire la classe syntaxique d'une phrase simple à partir du vecteur probabiliste, en utilisant les arbres de décision. Étant donné une grammaire G et une phrase S : $w_1, w_2, ...w_n$, nous avons montré comment trouver l'arbre de décomposition syntaxique préférable en utilisant le vecteur de la phrase. Toutes les prédictions sont approximatives.

8 Recherche et explications

Dans ce chapitre, nous analysons plusieurs flux de textes ou Twitter et utilisons leurs corrélations de contenu, introduites dans la section 5.4.2.3. La distance entre deux flux est l'inverse de leur corrélation, de même pour la distance entre deux clusters. Ces distances vont nous permettre de définir une distance *dist* entre deux mots, dans le contexte de plusieurs flux.

Nous introduisons alors le concept de *Recherche dans plusieurs flux* : c'est la généralisation d'un moteur de recherche, comme Google, dans le contexte de plusieurs flux dynamiques. Etant donnés des mots clés, $\sigma_1, \sigma_2, ..., \sigma_l$, on va rechercher les mots σ qui minimisent $\sum_{i=1,...,l} dist(\sigma, \sigma_i)$. Chaque solution σ aura une trace qui permettra de suivre les composantes géantes et les flux, qui sera le témoin du minimum : ces sont les explications.

A partir d'une matrice de distance entre les flux, nous pouvons utiliser les méthodes de phylogénie qui construisent un arbre de décomposition dont les feuilles sont les flux, avec des coûts sur les arêtes. Nous étendons cet arbre en un DAG (Directed Acyclic Graph) en plaçant un noeud pour chaque composante géante d'un flux et une arête entre le noeud de la composante et le noeud du flux. Nous plaçons ensuite un noeud pour chaque mot ou tag et une arête entre ce mot et chaque composante géante où ce mot se trouve. Les distances entre mots seront les plus courts chemins dans ce graphe.

Nous stockons les composantes géantes de chaque fenêtre, pour chaque flux, plus exactement le 2-core de chaque composante, comme un ensemble de nœuds et un ensemble d'arêtes. Chaque cluster est le résultat d'un tirage uniforme ou pondéré réalisé par un Réservoir qui nous permet cependant d'approximer les distances.

Dans la première section, nous rappelons la définition de la corrélation de flux. Dans la deuxième section, nous décrivons l'arbre phylogénique entre les flux. Dans la troisième section, nous étendons l'arbre en un DAG et définissons la distance entre deux mots, dans le contexte de flux. Dans la quatrième, nous présentons le moteur de recherche dans ce cadre. Dans la cinquième section, nous décrivons les explications disponibles pour les résultats de ce moteur.

8.1 Corrélation de flux

Nous avons introduit cette notion dans la section 5.4.2.3, pour estimer la corrélation entre deux flux Twitter. La notion se généralise aux flux de textes, puisque nous suivons la même approche. Pour un flux i et un graphe G_i associé, soit $C_{i,j}$ le j-ème (γ, δ)-cluster de C_i et

$$C_i = \bigcup_j C_{i,j}$$

l'ensemble des (γ, δ)-clusters du graphe G_i, pour $i = 1$ ou 2.

Definition 8. *La corrélation de deux flux définis par G_1, G_2 est le coefficient $\rho = J(C_1, C_2)$.*

Il est alors possible d'approximer la corrélation en utilisant les composantes géantes des graphes. Il suffit de considérer le 2-core de chaque composante géante. En effet, comme le montre [47], le 2-core de la composante géante du Réservoir est une bonne approximation, au sens ensembliste, de chaque $C_{i,j}$ et donc de l'union.

Dans la suite, nous utilisons plutôt la distance entre deux flux.

Definition 9. *La distance $dist(C_1, C_2)$ entre deux flux définis par G_1, G_2 est le coefficient $1 - \rho$.*

Ces définitions se généralisent aux graphes G_1, G_2 définis par des fenêtres dans les flux.

8.2 Phylogénie entre flux

Supposons que nous observions plusieurs flux. Les distances et corrélations vont varier au cours du temps. Pour la fenêtre à l'instant t, soit A_t la matrice de distance entre les flux i et j :

$$A_t(i, j) = dist(i, j)$$

La méthode Neighbor Joining [56], construit un arbre T avec des coûts sur les arêtes, de telle sorte que chaque flux apparaisse comme une feuille dans T et que la distance $d'(i, j)$ entre deux feuilles dans l'arbre soit approximativement la distance définie par la matrice A_t. Cette construction de l'arbre phylogénique suppose une propriété additive des distances, mais il existe toujours une solution approximative.

En quoi la phylogénie est-elle plus intéressante que la matrice de correlation ? Etant donné un flux i, i.e. une feuille de l'arbre, on peut trouver les flux proches plus efficacement en utilisant l'arbre phylogénique qu'en utilisant la matrice de distance. En effet on trouve un flux proche sans parcourir toute la matrice.

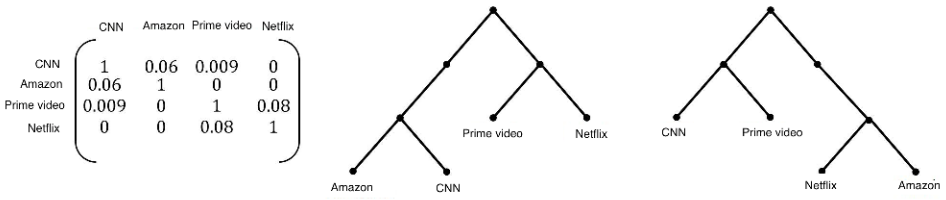

FIGURE 8.1: Arbre phylogénique T (centre) à partir de la matrice de distance, et un autre arbre proche T' (droite)

Avec la matrice 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 8.1. Étant donné un flux, les voisins d'une feuille de T sont en général les feuilles les plus proches dans T, et nous utilisons cette propriété dans la recherche par corrélation.

8.3 Distance entre mots, relative à des flux

La figure 8.2 indique comment compléter l'arbre phylogénique introduit à la section précédente 8.2 pour définir un DAG dont les nœuds représentent les flux, les clusters et les mots. Chaque flux est une feuille de l'arbre phylogénique : on place tous les clusters d'un même flux comme des nœuds connecté au flux par des arêtes de poids défini ci-dessous. Les mots d'un cluster sont des nœuds connectés aux clusters dont ils font partie, à un niveau encore inférieur, connectés par des arêtes de poids défini ci-dessous.

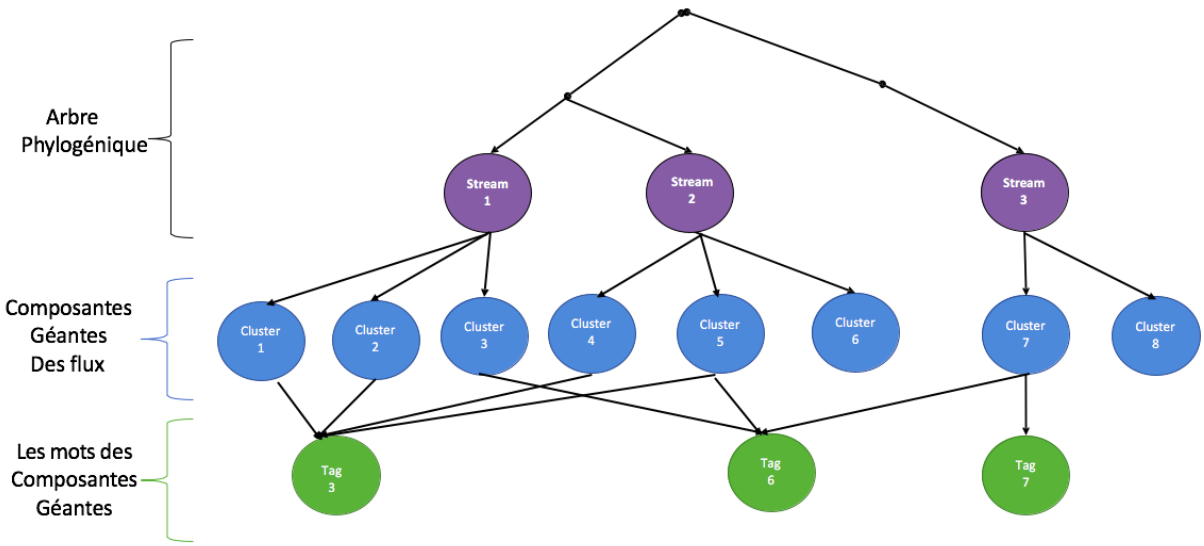

FIGURE 8.2: DAG de recherche qui regroupe les flux, les clusters et les mots.

On cherche des coûts sur les arêtes entre 0 et 1, qui privilégient les clusters le plus récents et qui tendent vers 1 quand l'âge du cluster tend vers l'infini. La fonction

$$f(t) = \frac{t}{t+2}$$

où $t = 0, 1, 2, ...$ est l'âge de la fenêtre. La fenêtre courante correspond à $t = 0$ et $f(t)$ vaut zéro. Pour $t = 1, 2, 3...$, les fenêtres précédentes (décalée de $t * \lambda$), la fonction $f(t)$ vaut $1/3, 1/2, 3/5...$ et tend vers 1 quand t croît.

Les coûts des arêtes du graphe de la figure 8.2 combinent deux critères : privilégier les clusters récents et les noeuds de grand degré. Ils sont définis comme suit :

- Le coût des arêtes de l'arbre phylogénique suit la construction de l'arbre phylogénique,
- Le coût des arêtes entre les noeuds de flux et les noeuds de clusters est de $f(t)$ si t est l'âge du cluster,
- Le coût entre les mots et les clusters est inversement proportionnel aux degrés d des nœuds dans le cluster et dépend de l'âge de cluster :

$$\frac{1}{d} + \frac{f(t)}{2}$$

Le coût $\mathsf{dist}_\mu(\sigma_1, \sigma_2)$ d'un chemin μ entre σ_1 et σ_2 est la somme des coûts des arêtes du chemin. Nous avons 3 cas possibles pour deux tags σ_1, σ_2, qui sont illustrés par la figure 8.2 :

- Cas 1 : σ_1, σ_2 sont dans le même flux et dans les mêmes clusters
- Cas 2 : σ_1, σ_2 sont dans le même flux mais dans deux clusters différents
- Cas 3 : σ_1, σ_2 sont dans deux flux différents et dans deux clusters différents

Les tags σ_1 et σ_2 de degrés d_1 et d_2 appartiennent à des composantes géantes, C_{1,t_i} et C_{2,t_j} dans des flux f_1 et f_2. Soit d_T la distance entre f_1 et f_2 dans l'arbre phylogénique. Si on fixe un chemin μ entre σ_1 et σ_2 la distance est :

$$\mathsf{dist}_\mu(\sigma_1, \sigma_2) = \begin{cases} \frac{1}{d_1} + \frac{1}{d_2} + f(t) & \text{si } C_{1,t_i} = C_{1,t_j} \wedge f_1 = f_2 \\ \frac{1}{d_1} + \frac{1}{d_2} + 3/2 * f(t_i) + 3/2 * f(t_j) & \text{si } C_{1,t_i} \neq C_{1,t_j} \wedge f_1 = f_2 \\ \frac{1}{d_1} + \frac{1}{d_2} + 3/2 * f(t_i) + 3/2 * f(t_j) + d_T & \text{si } C_{1,t_i} \neq C_{1,t_j} \wedge f_1 \neq f_2 \end{cases}$$

La distance entre deux tags σ_1 et σ_2 est :

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

Cette définition se généralise à deux nœuds arbitraires du DAG. Nous pouvons vérifier que la fonction dist est une fonction binaire symétrique qui satisfait l'inégalité triangulaire.

8.4 Moteur de recherche par similarité

Nous présentons un algorithme de recherche pour trouver des tags proches d'une liste de tags $\sigma_1, ...\sigma_l$.

Recherche par similarité $(\sigma_1, ...\sigma_l)$:

- Trouver les mots σ qui minimisent :

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

Etant donné un tag σ, on peut calculer par l'algorithme de Dijkstra les mots les plus proches et les plus courts chemins vers $\sigma_1, \sigma_2, ..., \sigma_l$ et donc $\sum_{i=1,...,l} dist(\sigma, \sigma_i)$. Nous utilisons cette procédure comme une boite noire.

8.4.1 Algorithme de recherche

Algorithme de recherche$(\sigma_1, ...\sigma_l)$:

— Pour chaque mot (σ_i), $i = 1, ...l$, soit C_{i,t_i} le cluster le plus récent qui contient σ_i,

— Soit $\sigma_{d_{max,i}}$ le nœud de plus grand degré (le centre) de la composante C_{i,t_i},

— Pour chaque $\sigma_{d_{max,i}}$, on calcule $\sum_{j=1,...,l} dist(\sigma_{d_{max,i}}, \sigma_j)$ à l'aide de la boite noire,

— Les réponses sont les $\sigma_{d_{max,i}}$ qui minimisent la somme précédente.

Notons que si σ est une solution optimale, ce n'est pas nécessairement le centre d'une composante. En effet, σ est dans un cluster C mais son centre c n'est pas nécessairement la solution minimum. La somme $\sum_{j=1,...,l} dist(c, \sigma_j)$ peut être supérieure à la somme $\sum_{j=1,...,l} dist(\sigma, \sigma\_j)$, selon les positions des arêtes du centre vers les clusters C_{i,t_i}.

Nous allons montrer que les solutions $\sigma_{d_{max,i}}$ sont des solutions approchées dans le sens où $\sum_{j=1,...,l} dist(\sigma_{d_{max,i^*}}, \sigma_j)$ est plus petit que l'optimum multiplié par une constante c, par exemple $c = 2$.

8.4.2 Justification de l'algorithme de recherche

Etant donné un tag σ, on peut calculer par l'algorithme de Dijkstra, aussi appelé $A*$, les plus courts chemins vers $\sigma_1, \sigma_2, ..., \sigma_l$ et donc $\sum_{i=1,...,l} dist(\sigma, \sigma_i)$. Il existe trois types de chemin μ entre deux tags :

• Chemins de type 1 : tag-cluster-tag-cluster-tag : $\sigma_1, C_1, \sigma_2, C_2, \sigma_3$, avec σ_2 un noeud intermédiaire. Pour un chemin de type 1, soit d'_1 le degré de σ_2 dans C_1 et d_2 le degré de σ_2 dans C_2, et soit d'_2 le degré de σ_3 dans C_2. Le coût total est de :

$$f(t_1)/2 + 1/d_1 + f(t_1)/2 + 1/d'_1 + f(t_2)/2 + 1/d_2 + f(t_2)/2 + 1/d'_2$$
$$= f(t_1) + 1/d_1 + 1/d'_1 + f(t_2) + 1/d_2 + 1/d'_2$$

En général pour un chemin qui a $i - 1$ noeuds intermédiaires, le coût sera de :

$$\sum_i f(t_i) + \sum_i (1/d_i + 1/d'_i)$$

• Chemins de type 2 : tag-cluster-stream-cluster-tag $\sigma_1, C_1, s, C_2, \sigma_2$ pour des clusters du même stream s. Le coût est de :

$$f(t_1)/2 + 1/d_1 + f(t_1) + f(t_2) + 1/d_2 + f(t_2)/2$$
$$= 3/2.(f(t_1) + f(t_2)) + (1/d_1 + 1/d_2)$$

• Chemins de type 3 : tag-cluster-stream-philogénie-stream-cluster-tag, $\sigma_1, C_1, s_1, T, s_2, C_2, \sigma_2$, pour deux flux s_1, s_2 et T l'arbre phylogénique. Le coût est de :

$$f(t_1)/2 + 1/d_1 + f(t_1) + f(t_2) + 1/d_2 + f(t_2)/2 + d_T(s_1, s_2)$$
$$= 3/2.(f(t_1) + f(t_2)) + (1/d_1 + 1/d_2) + d_T(s_1, s_2)$$

Le coût générique entre deux tags σ_1 et σ_2 est le chemin de type 2 ou 3 où on prend les clusters les plus récents. Pour un chemin de type 1, la somme $\sum_i f(t_i)$ croît comme la longueur chemin et dépassera la coût d'un chemin de type 2 ou 3.

Lemme 5. *Si un chemin μ de type 1 entre deux tags σ_1 et σ_2 est meilleur que le chemin générique, alors il est court.*

Démonstration. Un chemin de type 1 de longueur 6, avec deux noeuds intermédiaires a un coût de :

$$\sum_{i=1,2,3} f(t_i) + \sum_{i=1,2,3} (1/d_i + 1/d'_i)$$

Pour un modèle où tous les $f(t_i)$ ont une même moyenne, le chemin générique de type 2 a un coût inférieur, $3/2.(f(t_1) + f(t_2)) + (1/d_1 + 1/d_2)$, d'un facteur environ 1/2. Pour un chemin de type 3, le coût additionnel $d_T(s_1, s_2)$ est constant. Il existe donc une longueur $2.p$ telle que le coût d'un chemin de type 1 de longueur $2.p$ sera supérieur au coût du chemin de type 3. □

Soit σ l'optimum, qui minimise $\sum_{i=1,...,l} dist(\sigma, \sigma_i)$ et soit ρ^* la valeur minimale. Montrons que les solutions produites par l'algorithme ne sont pas très éloignées de ρ^*. L'algorithme produit les $\sigma_{d_{max,i}}$, les centres des composantes les plus récentes des σ_i. Soit i^* l'indice qui minimise

$$\sum_{j=1,...,l} dist(\sigma_{d_{max,i}}, \sigma_j)$$

Lemme 6. *Il existe une constante c telle que $\sum_{j=1,...,l} dist(\sigma_{d_{max,i^*}}, \sigma_j) \leq c.\rho^*$.*

Démonstration. Montrons que la propriété est vraie pour chaque i et donc aussi pour i^*. Pour chaque i, $dist(\sigma_{d_{max,i}}, \sigma_i) \leq dist(\sigma, \sigma_i)$ car $d_{max,i}$ est le point le plus proche de σ_i : il appartient au même cluster et a un degré maximum. Pour les autres σ_j, où $j \neq i$, on applique l'inégalité triangulaire, $dist(\sigma_{d_{max,i}}, \sigma_j) \leq dist(\sigma_{d_{max,i}}, \sigma) + dist(\sigma, \sigma_j)$. D'après le lemme 5, il existe une constante c telle que :

$$dist(\sigma_{d_{max,i}}, \sigma_j) \leq c.dist(\sigma, \sigma_j)$$

et on obtient bien le résultat annoncé.

□

8.4.3 Structures de données

Ce qui est stocké dans le temps. A 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, timestamp, list(high-degree nodes), list(nodes(nodes,degree)))))* est la table qui fournit la liste des noeuds de haut degré et la liste des noeuds avec leur degré, dans un cluster donné,

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

8.5 Explications

L'algorithme de recherche prend le centre c_i des clusters de chacun des tags, puis regarde le meilleur des centres, celui qui a la distance la plus courte par-rapport à tous les autres tags. Chacun des centres va appliquer l'algorithme de *Dijkstra* pour calculer la somme de distance qui minimise la fonction de coût $\sum_{j=1,\dots,l} dist(\sigma_{d_{max,i^*}}, \sigma_j)$. On obtient via l'algorithme de Dijkstra les chemins qui vont être de type 1 ou de type 2 ou 3. Les chemins de type 1 ne peuvent pas être très longs. Par défaut l'algorithme nous donne le chemin de type 2. Nous observons trois chemins, comme le montre la figure 8.3. À partir du centre c nous obtenons deux chemins de type 1 et un chemin de type 2. Nous avons trouvé les plus courts chemins de cette manière. Ce tag est une réponse intéressante aus questions de trois tags, parce qu'il a un indice pour cette fonction d'optimum qui n'est pas mauvais.

L'algorithme de recherche trouve le centre c_i du cluster le plus récent de chacun des tags, qui a la distance totale aux autres tags la plus courte. Chacun des centres applique l'algorithme de *Dijkstra* et estime la somme des distances

$$\sum_{j=1,\dots,l} dist(\sigma_{d_{max,i^*}}, \sigma_j).$$

Prenons par exemple une recherche sur les trois tags *CNN, AMAZON, Prime video* qui sont appelés *tag* 3, *tag* 6, *tag* 7 sur la figure 8.3. La solution obtenue est le centre du cluster 5 et les trois chemins trouvés par *Dijkstra* sont en rouge bleu et vert.

L'algorithme de Dijkstra trouve des chemins de type 1 ou de type 2 ou 3. Les chemins de type 1 ne peuvent pas être très longs. La figure 8.3 montre trois plus courts chemins. À partir du centre c nous obtenons deux chemins de type 1 et un chemin de type 2. La solution n'est pas l'optimum garanti, mais est dans un rapport fixe à l'optimum, comme le montre le lemme 8.4.2.

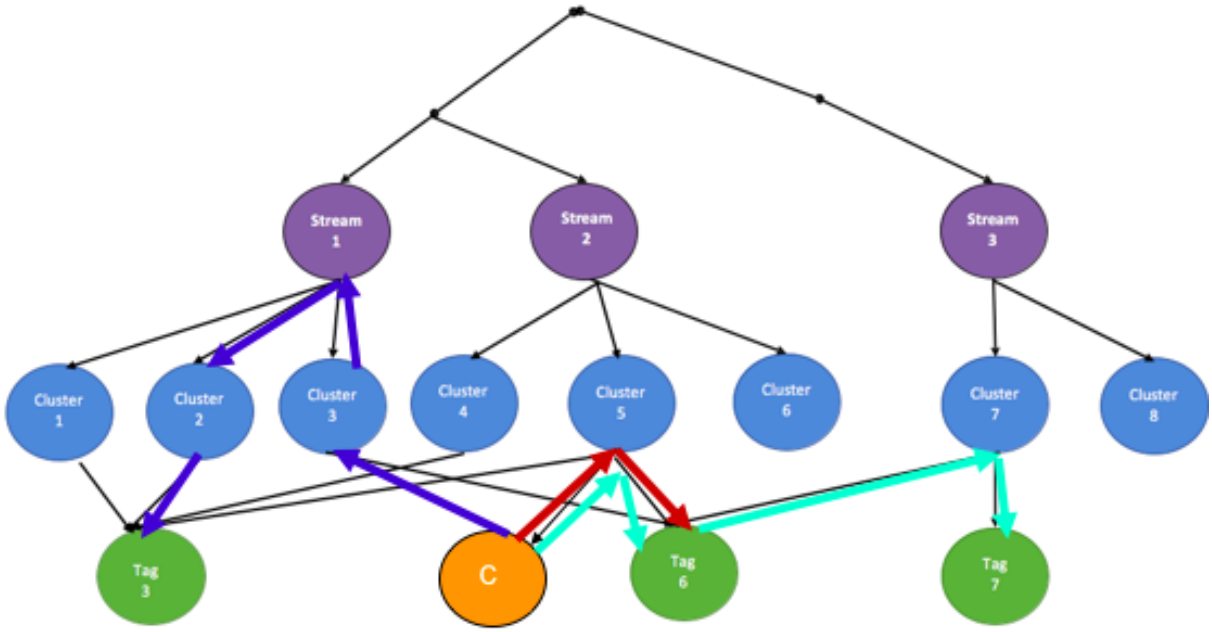

FIGURE 8.3: DAG de recherche.

8.6 Conclusion

La corrélation de contenus entre deux composantes définit une distance qui se généralise aux flux. On peut donc associer un arbre phylogénique aux flux, puis l'étendre aux composantes géantes et aux tags. Nous avons introduit la notion de recherche par corrélation basée sur ces distances. Ces travaux conjuguant plusieurs domaines, ouvrent des champs nouveaux d'applications.

9 Expériences

Nous présentons dans ce chapitre les différentes expériences réalisées dans le cadre de cette thèse.

Dans la première section, nous présentons les expériences dans le cadre d'un flux Twitter. Dans la deuxième section, nous montrons des expériences pour un flux de texte, en comparant l'échantillonnage uniforme avec l'échantillonnage pondéré. Dans la troisième section, nous présentons des expériences sur le chapitre recherche et explication.

9.1 Flux Twitter

Aujourd'hui, Twitter présente les tendances à un instant donné, en affichant les tags les plus fréquents. Nous avons étendu un programme présenté par [61], pour estimer les corrélations entre les tags et nous l'étendons pour les corrélations entre flux. Cela permet de voir comment les tags les plus fréquents sont reliés entre eux, de connaître la fréquence des paires de tags.

Il s'agit d'un programme Python [1], exécutable comme un notebook sur *Google Colab*. La première version nécessite un compte de développeur Twitter [2] qui spécifie 4 clés. Le code python doit inclure les bonnes clés et lit les entrées suivantes qui spécifient les paramètres importants ainsi que ceux des fenêtres dynamique introduites dans la section 5.4.2.2 :

1. La taille du Réservoir (k),
2. Les mots-clés de Twitter,
3. La longueur τ de la fenêtre,
4. Le paramètre de pas λ,
5. La valeur seuil, la taille minimale des composantes géantes,
6. La durée de l'observation.

La sortie est un ensemble de fichiers, un pour chaque fenêtre. Chaque fichier contient des arêtes $(u, v, texte)$ où u, v sont des nœuds et le texte est le contenu du tweet. Par exemple, Le "@thomasjacksonjr" a envoyé un tweet qui est le texte ci-dessous, dans laquelle $v = \#PrimeVideo$ est un des 4 tags et l'arête :

($@thomasjacksonjr$, $\#PrimeVideo$, "Watched $\#ABCMurders$ from beginning to end and it was pure #mystery and delight. Excellent show and a must watch for $\#mysterylovers$. Great show on $\#PrimeVideo$.").

1. https ://github.com/alassou/t/blob/master/topic.ipynb
2. https ://developer.twitter.com

9.1.1 Expérience sur un flux

Une exécution possible précise : $k = 400$, le mot clé : "CNN", $\tau = 12min$, $\lambda = 3min$, seuil $= 10$, la durée de l'observation $= 24min$. Ces paramètres définissent 5 fenêtres qui se chevauchent et la sortie contient 5 fichiers.

Nous observons 3.10^3 arêtes, générés par 10^3 tweets par minute pendant $24min$. La figure 9.1 décrit un Réservoir de 400 d'arêtes où tous les petites composantes (de taille inférieure au seuil=10) ont été enlevées. Pour une observation de $\tau = 12$, nous avons approximativement $m = 3.10^4$ et $n = 8000$, comme $m = O(n.\log n)$. La taille k du Réservoir est $(\sqrt{n}.\log n) = 90.4 = 360$, soit moins de 400 dans l'expérience.

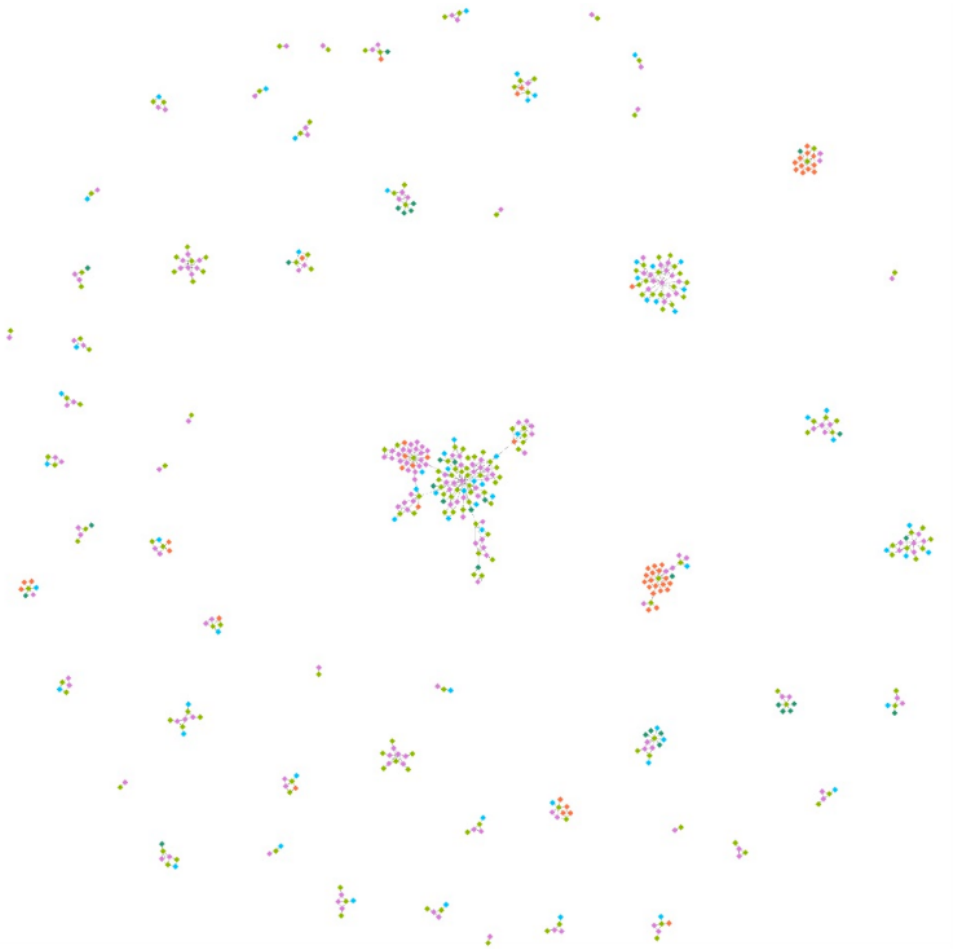

FIGURE 9.1: Les composantes géantes du Réservoir

9.1.2 Expérience sur plusieurs flux

Supposons une autre expérience avec 4 flux twitter sur les tags #CNN, #FoxNews, #Bitcoin, et #Ripple pendant 24 heures avec un taille de fenêtre $\tau = 1h$ et un intervalle de temps de $\lambda = 30mins$. La figure 9.3 indique le nombre d'arêtes dans une fenêtre, environ $m = 30.10^3$ par flux, pour chaque flux, donc 10^6 d'arêtes en 24 heures. Pour le #Bitcoin avec $k = 400$, la taille de la composante géante est d'environ 100.

Ces données influencent le développement d'un moteur de recherche par similarité, présenté dans la section 8.2.

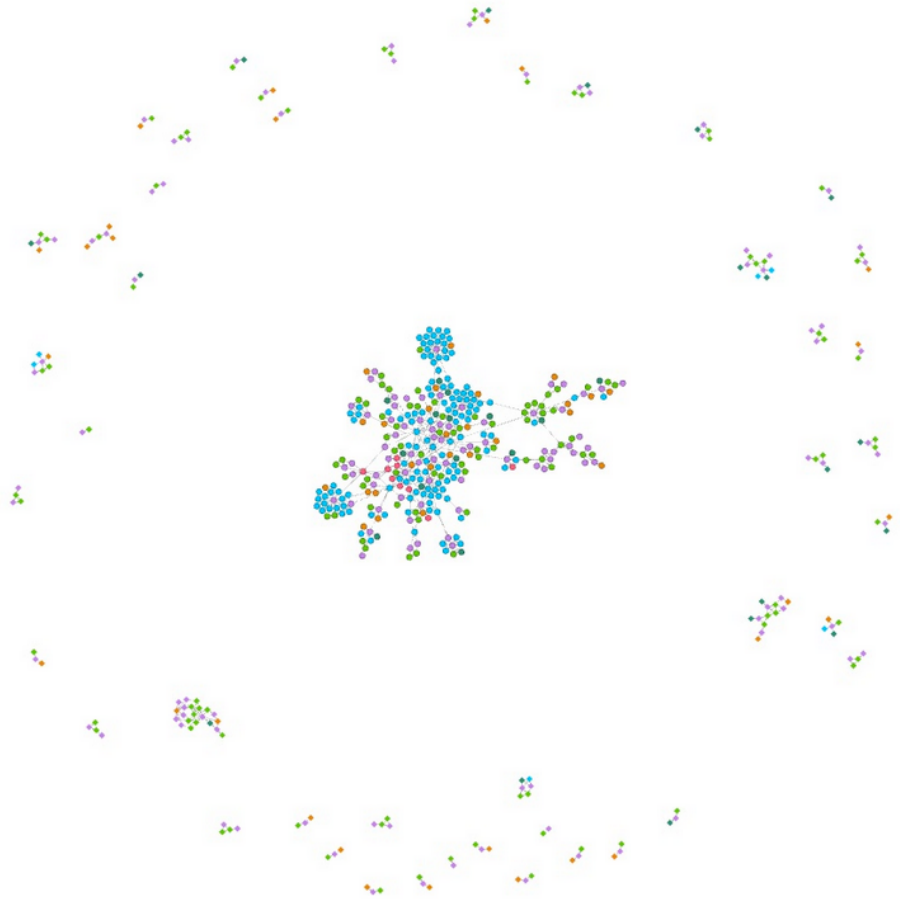

FIGURE 9.2: Les composantes géantes du Réservoir pour le mot clé *Bitcoin*.

9.2 Analyse de texte

Nous pouvons appliquer notre méthode à un corpus de textes standards. Pour chaque phrase, nous appliquons d'abord *la lemmatisation*, *la reconnaissance des entités* [49] et éliminons les *stop words* [3] et les mots les moins fréquents. Nous générons soit les *bigrammes* (paires de mots contigus) [50] soit toutes les paires possibles d'une phrase donnée, comme des arêtes potentielles. Pour l'échantillonnage uniforme, les poids des arêtes sont constants (par exemple égal à 1). Pour l'échantillonnage pondéré, le poids d'une arête est la similarité entre deux mots de *Word2vec* présentée dans la section 6.2. Dans les deux cas, nous traitons le texte comme un flux.

Nous évaluons nos méthodes sur l'ensemble de données *(NIPS)*[4], *(Googleplaystore_user_reviews)*

3. stop word est un mot commun qu'il est inutile d'indexer ou de l'utiliser dans une recherche. En français, par exemple « le », « la », « de », « du », « ce »...

4. https ://www.kaggle.com/benhamner/nips-paperspapers.csv

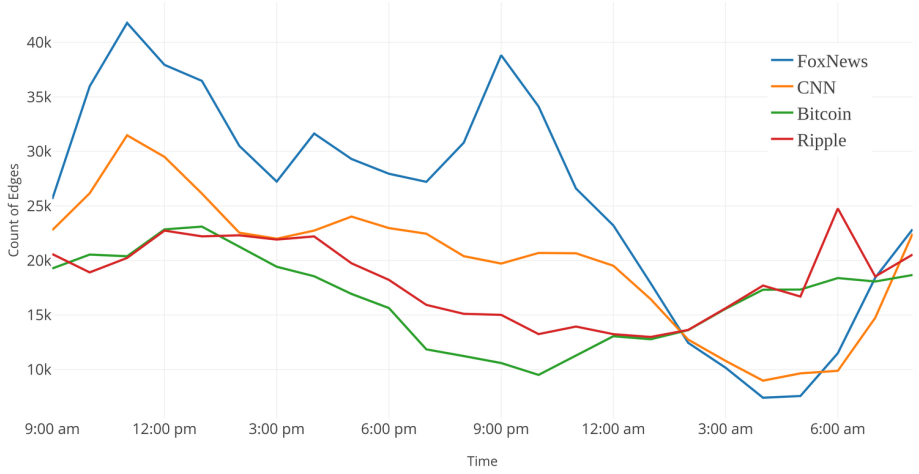

FIGURE 9.3: Nombre d'arêtes/par heure pour 4 flux pendant 24h

dataset[5] et le corpus de *Stanford Natural Language Inference (SNLI)* [13] qui consiste en une collection de phrases en anglais. Nous utilisons uniquement le jeu de données *test*[6], qui contient 3000 phrases. Nous avons obtenu 71269 de paires de mots.

9.2.1 Échantillonnage uniforme ou pondéré

Nous appliquons l'échantillonnage uniforme comme dans le cas de Twitter ou pondéré où le poids est la valeur absolue de la similarité *Word2vec* entre deux mots. Nous introduisons une notion de stabilité pour la construction du Réservoir introduite dans la section 6.3. Supposons que nous effectuons deux passages indépendants sur le corpus D, avec deux Réservoirs de même taille k. Si nous observons une composante géante dans chaque Réservoir, c'est-à-dire C_1 pour la première passe et C_2 pour la deuxième passe, quelle est la taille de l'intersection $C_1 \cap C_2$ par rapport à la taille de C_1 ou de C_2 ?

Pour les figures [9.4,9.5] et le tableau 9.1, nous regardons les *bigrammes*, tandis que pour la figure 9.6 nous regardons toutes les paires possibles. Pour l'échantillonnage pondéré, la figure 9.4 donne la taille de la composante géante en fonction de k.

En regardant les *bigrammes* sur une phrase donnée, la figure 9.5 montre la comparaison entre $\rho(D,k)$ pour le Réservoir uniforme et le Réservoir pondéré avec différentes tailles de Réservoir $k = 100, 200, ...500$. Nous avons observé que dans le cas du Réservoir uniforme, $\rho(D,k)$ augmente avec la taille du Réservoir et dans le cas du Réservoir pondéré, $\rho(D,k)$ est presque stable comme le montre la figure 9.5. Dans tous les cas, le Réservoir pondéré est toujours plus stable.

Nous lisons trois différents benchmarks et appliquons l'échantillonnage pondéré du Réservoir.

5. https ://www.kaggle.com/lava18/google-play-store-appsgoogleplaystore_user_reviews.csv
6. https ://github.com/alassou/t/blob/master/snli.csv

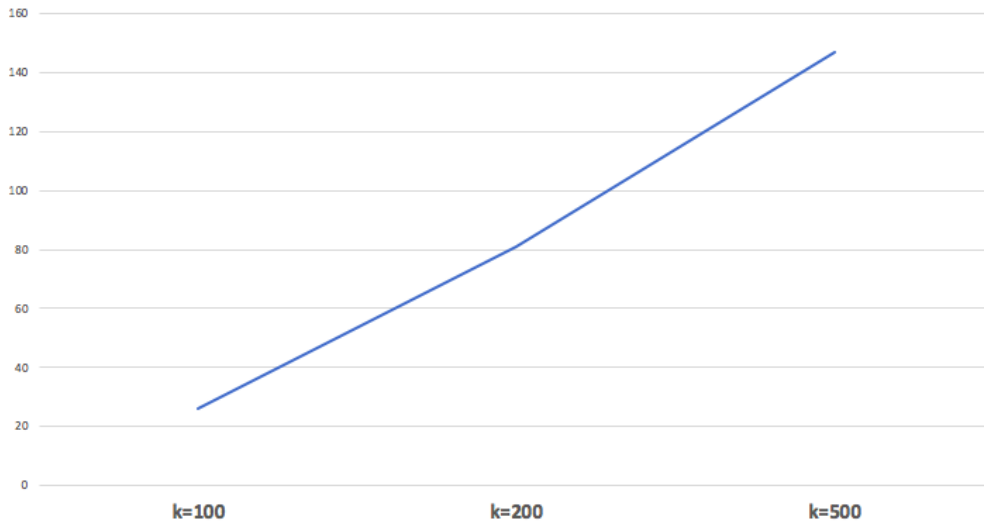

FIGURE 9.4: Taille de la composante géante pour le Réservoir pondéré

Le tableau 9.1 montre la comparaison de $\rho(D, k)$ pour les trois différents benchmarks et pour des différentes valeurs de k.

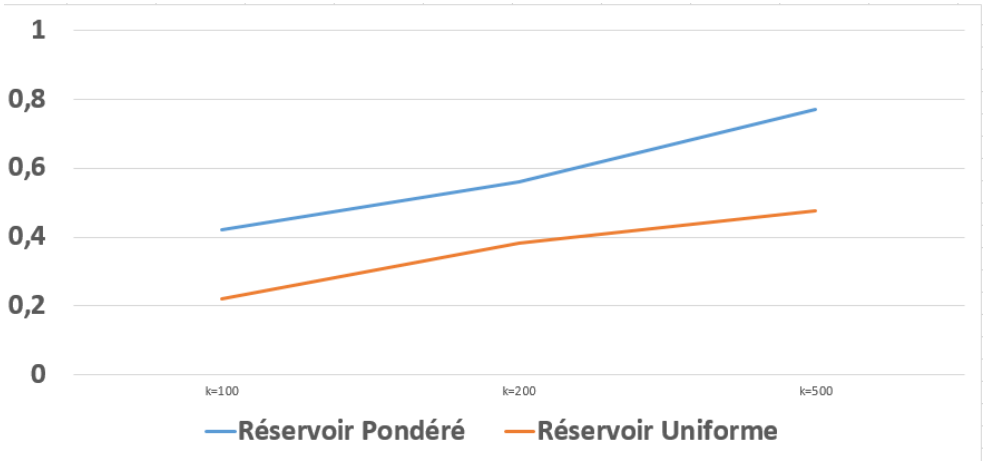

FIGURE 9.5: Stabilité du Reservoir sampling et de l'échantillonnage pondéré.

Benchemarks	**k=100**	**k=200**	**k=500**
SNLI	0,42	0,56	0,77
NIPS PAPERS	0,58	0,65	0,72
Googleplaystore_user_reviews	0,15	0,52	0,61

TABLE 9.1: Stabilité du Réservoir pondéré

La figure 9.6 montre la comparaison de stabilité entre l'échantillonnage uniforme et pondéré, en regardant tous les paires possibles de la phrase avec différentes tailles de Réservoir $k = 100, 200, ...500$.

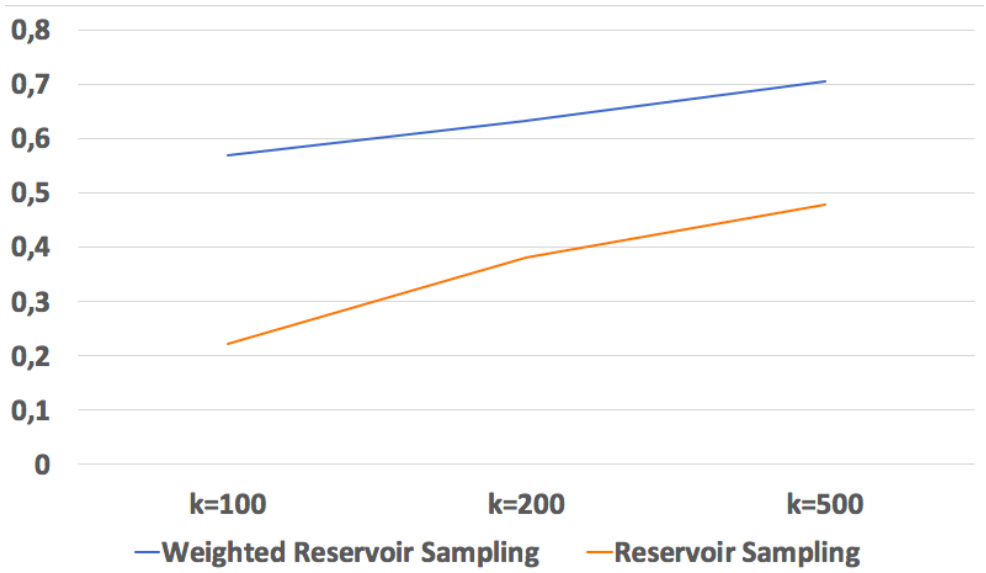

FIGURE 9.6: Stabilité du Reservoir sampling et de l'échantillonnage pondéré sur tous les paires.

9.2.2 Classification

Des mesures de qualité [64] ont été introduites pour comparer les différentes méthodes de classifications présentées dans la section 6.4. Dans notre cas, chaque expérience fournit un ensemble de composantes géantes et chaque composante géante peut être interprétée comme un sujet. La composante géante n'est pas une distribution mais un graphe avec sa propre structure.

Nous pouvons comparer un support de distributions *LDA* présentées dans la section 3.3.2 avec l'ensemble des noeuds du graphe de la composante géante. Comme nous répétons différentes expériences, l'espérance de la stabilité est très similaire à la stabilité du sujet introduit dans [64]. Une différence importante est que *LDA* doit accéder à toutes les données, alors que nous n'accédons qu'aux données en streaming en faisant 1 seule passe.

Nous avons comparé notre méthode avec celle de *LDA* pour $k = 2$.

Topic1 a été décrit par les mots significatifs suivants : ['man', 'woman', 'wear', 'boy', 'walk', 'girl', 'dog', 'stand', 'street', 'play', 'child', 'dress', 'hold', 'water'], tandis que Topic2 a été décrit par les mots significatifs suivants ['man', 'sit', 'people', 'woman', 'look', 'stand', 'group', 'shirt', 'wear', 'table', 'hold', 'player', 'work', 'play', 'ball']. Les distributions sont données dans la figure 9.7.

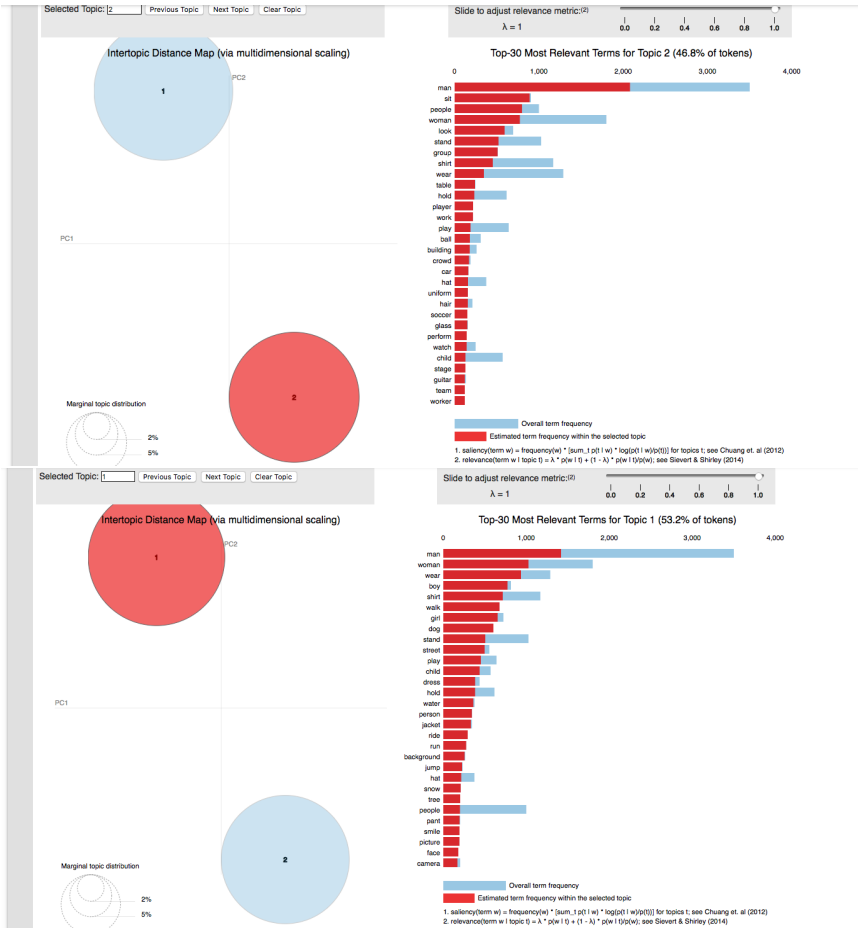

FIGURE 9.7: Distributions de 2 topics

Si nous prenons le Réservoir pondéré stable, quelle partie de ces sujets couvrons-nous comme noeuds ? Cela est indiqué dans la figure 9.8. Plus k augmente, plus nous couvrons les mots du Topic, pour atteindre 100% pour k= 500.

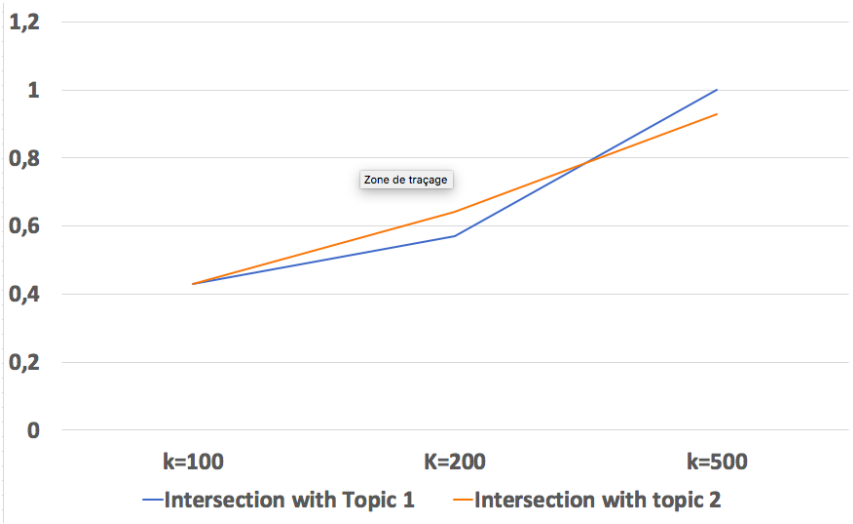

FIGURE 9.8: Fraction des thèmes de LDA comme des noeuds de la composante géante du réservoir pondéré

Dans le cas général, si on prend le Réservoir pondéré dans des fenêtres, on va avoir un certain nombre de composantes géantes pour lesquelles on a des distances. Si on applique l'algorithme $k - means$, on aura $kclasses$. Finalement, on compare le représentant de chacune des classes avec les topics de LDA. Dans le cas particulier représenté ci-dessus, on a une seule fenêtre et on observe deux composantes géantes.

9.3 Décompositions hiérarchiques des composantes géantes

Chaque composante géante est le $2 - Core$ d'un graphe détaillé dans la section 2.3.4.1. Nous pouvons explorer ce graphe en commençant par les nœuds de degré maximum et itérer jusqu'à ce que nous explorions l'ensemble de la composante géante. Avec le graphe de la figure 9.10, nous obtenons la décomposition en arbre de la figure 9.11. Nous commençons avec le nœud 1 de degré maximum et nous explorons tous les nœuds à distance 1. À l'étape suivante, le nœud 6 est de degré maximum et nous obtenons le nœud 3. La décomposition de l'arbre est de profondeur 2.

Dans la figure 9.12, chaque nœud est un tag et le texte du tweet est associé à chaque arête. Nous ne montrons que le texte de l'arête e_1.

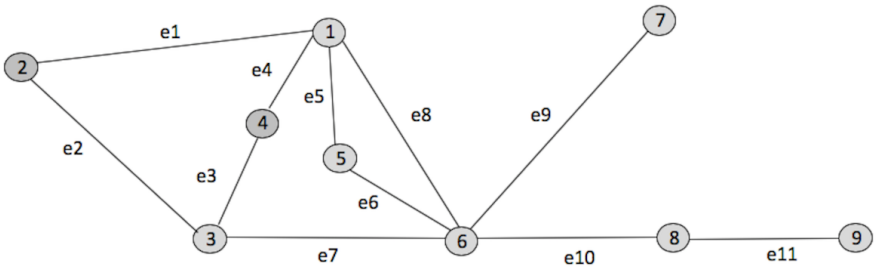

FIGURE 9.9: La grande composante connectée (C) dans le Réservoir

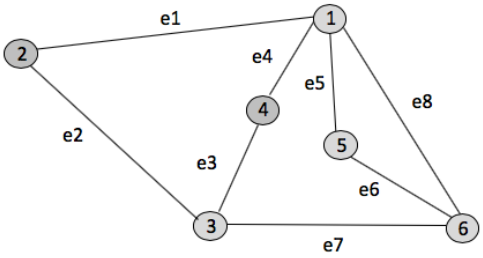

FIGURE 9.10: Le $2 - core(C)$

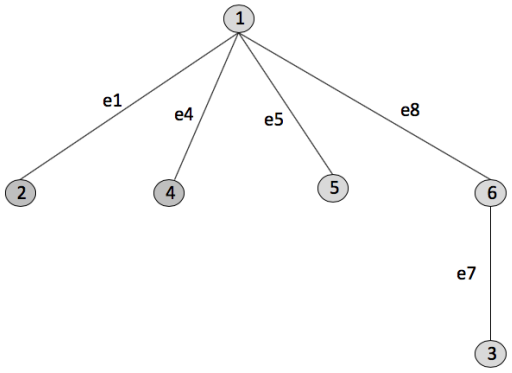

FIGURE 9.11: Arbre de décomposition

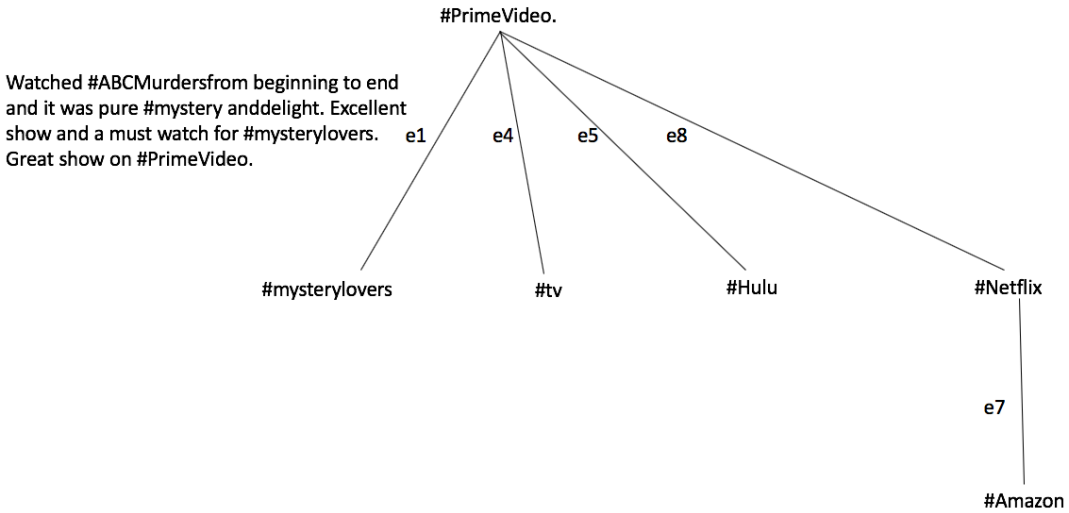

FIGURE 9.12: Arbre de décomposition où les noeuds sont des tags

10 Conclusion

L'objectif de cette thèse est de montrer comment approcher les propriétés de trois différents types de données (entrepôts de données au sein d'une base de données, graphes issus des réseaux sociaux, texte) à partir d'échantillons. Les résultats de cette thèse se décomposent en deux grandes parties. La première partie traite de l'échantillonnage de données afin d'approximer certaines propriétés et la deuxième partie approfondit un modèle morphologique de texte pour enrichir l'analyse du texte.

La première partie de la thèse couvre les sujets suivants :

- Dans les bases de données, nous avons montré comment approximer des requêtes OLAP à l'aide d'un tirage non uniforme. Nous avons également introduit une notion de préférence comme une distribution sur les *Mesures*. Nous avons montré comment nous pouvons généraliser cette méthode d'approximation, pour toute préférence, c'est-à-dire toute distribution sur les *Mesures*.

- Pour un flux de Tweets, nous construisons un flux d'arêtes qui représente le graphe Twitter G. Nous avons montré que l'échantillonnage uniforme des arêtes reflète les grands clusters de G. L'échantillonnage uniforme est stable pour Twitter.

- Pour un flux de textes, nous construisons un graphe dont les arêtes sont les paires de mots de la même phrase. Nous avons montré que l'échantillonnage pondéré est stable pour analyser les composantes géantes de ce graphe. Chaque composante géante définit des mots centraux et des phrases centrales. Nous pouvons classifier les composantes avec l'algorithme *k-means*.

- On introduit la notion de moteur de recherche à partir des données d'échantillonnage pour donner des réponses à des requêtes définis par des mots clés. Les réponses sont obtenues par corrélation entre les composantes géantes et les échantillons. Pour chaque réponse, le moteur propose des explications.

La deuxième partie de la thèse introduit un modèle statistique pour la morphologie des langues naturelles, inspiré par *Amis*, une langue naturelle qui a une riche morphologie. Nous avons construit les distributions classiques des préfixes, racines et suffixes les plus fréquents, et les distributions correspondantes avec une racine, un préfixe ou un suffixe possible. A partir des seconds moments des distributions, nous avons construit des vecteurs pour les préfixes, les racines et les suffixes qui capturent leurs corrélations. La morphologie d'un mot w est la structure

préfixe-racine-suffixe . On construit des vecteurs : $v_p(pre)$ pour le préfixe *pre*, $v_r(root)$ pour la racine *root* et $v_s(suf)$ pour le suffixe *suf*. Le vecteur du mot $v(w)$ est la concaténation des trois vecteurs. Nous avons défini un vecteur probabiliste associé à une phrase. L'analyse des préfixes et suffixes, détermine la plupart des composantes du vecteur de la phrase S, de manière probabiliste. Nous avons montré comment prédire la classe syntaxique d'une phrase simple à l'aide d'un arbre de décision. Étant donné une grammaire G et une phrase $S : w_1, w_2, ...w_n$, nous avons montré comment trouver l'arbre de décomposition syntaxique préférable en utilisant le vecteur de la phrase. Toutes les prédictions sont approximatives.

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

- Peut-on améliorer l'échantillonnage uniforme dans le cas des réseaux sociaux ? Nous n'utilisons pas d'historique sur les tags et utilisons la distribution uniforme par défaut. Nous pourrions construire la matrice de corrélation des tags, puis des vecteurs *Word2vec* comme dans le cas du langage naturel. Nous aurions alors un échantillonnage pondéré possible pour les paires de tags.
- Comment utiliser des vecteurs *Word2vec* structurés, comme ceux introduits pour l'analyse morphologique en ayant plusieurs distributions possibles sur lesquelles on peut faire de l'échantillonnage ? Les distributions des préfixes, racines et suffixes et leurs projections permettraient d'obtenir des échantillons plus variés.
- Comment classifier plusieurs flux à l'aide de l'algorithme *k-means*, appliqué aux composantes géantes de chaque fenêtre coulissante ? Plusieurs distances sont possibles entre les composantes, présentées au chapitre 6.4.1.
- Dans le domaine du *Question Answering*, comment utiliser les vecteurs de phrase basés sur la morphologie, introduits dans le chapitre 7 ? L'échantillonnage n'est pas adapté dans ce cadre, mais les vecteurs probabilistes associés aux phrases et les vecteurs *Word2vec* associés aux mots, donnent une information importante pour interpréter une question. Le mécanisme d'attention, basé sur les distributions de similarité entre un mot et les autres mots d'une phrase est à l'origine des techniques de *Transformers* utilisées pour la traduction d'une langue vers une autre. Comment traduire une question en langage naturel dans le contexte d'un schéma relationnel, en une requête SQL ?

  Dans le cas d'un ensemble de phrases, il peut être délicat d'isoler la question précise, en particulier en l'absence du symbole " ?". Les vecteurs de phrase capturent les actes de langage de manière approchée et pourraient contribuer à mieux appréhender les questions. Un *indice de compréhension* pourrait alors mesurer expérimentalement la qualité des réponses sur des ensembles de tests.

Bibliographie

[1] Himmelmann N. (Ed.). Adelaar, K. (Ed.). *The Austronesian Languages of Asia and Madagascar.* London : Routledge,, 2005.

[2] 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.

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

[4] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. A latent variable model approach to PMI-based word embeddings. *Transactions of the Association for Computational Linguistics*, 4 :385–399, 2016.

[5] Bogdan Babych and Anthony Hartley. Improving machine translation quality with automatic named entity recognition. In *Proceedings of the 7th International EAMT workshop on MT and other language technology tools, Improving MT through other language technology tools, Resource and tools for building MT at EACL 2003*, 2003.

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

[7] Y. Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. *Neural Probabilistic Language Models*, volume 3, pages 137–186. 05 2006.

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

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

[10] D. M. Blei, A. Y. Ng, and M. I. Jordan. Lattent dirichlet allocation. *Journal of Machine Learning Research*, 3 :993-1022, 2003.

[11] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.

[12] Bela Bollobas. *Graph theory : an introductory course / Bela Bollobas.* Springer Verlag,, 1979.

[13] Samuel Bowman, Gabor Angeli, Christopher Potts, and Christoper Manning. A large annotated corpus for learning natural language inference. *CoRR*, 08 2015.

[14] Isabelle Bril. Roots and stems : Lexical and functional flexibility in Amis and Nêlêmwa. *Studies in Language*, 41(2) :358–407, 2017.

[15] Isabelle Bril. Roots and stems : Lexical and functional flexibility in amis and nêlêmwa, 2017.

[16] Isabelle Bril, Achraf Lassoued, and Michel de Rougemont. A statistical model for morphology inspired by the amis language. In *Proceedings of Language, Ontology, Terminology and Knowledge Structures Workshop (LOTKS 2017)*, Montpellier, France, 2017. Association for Computational Linguistics.

[17] Kris Cao and Marek Rei. A joint model for word embedding and word morphology. *CoRR*, abs/1606.02601, 2016.

[18] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750. Association for Computational Linguistics, October 2014.

[19] Teresa Chen. Verbal constructions and verbal classifications in nataoran-amis. In *Series C. Canberra : Pacific linguistics*, 1987.

[20] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Comput. Linguist.*, 16(1) :22–29, March 1990.

[21] Ronan Collobert and Jason Weston. A unified architecture for natural language processing : deep neural networks with multitask learning. In *ICML '08*, 2008.

[22] M. de Rougemont and G. Vimont. The content correlation of streaming edges. In *IEEE International Conference on Big Data*, pages 1101–1106, 2018.

[23] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. 41(6) :391–407, 1990.

[24] Steve Cassidyi Diego Mollá, Menno van Zaanen. Named entity recognition in question answering of speech data. *Proceedings of the Australasian Language Technology Workshop*, pages 57–65, 2007.

[25] 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.

[26] 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, 2015.

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

[28] P. Erdős and A Rényi. On the evolution of random graphs. pages 17–61, 1960.

[29] Hossein Esfandiari, MohammadTaghi Hajiaghayi, and David P. Woodruff. Brief announcement : Applications of uniform sampling : Densest subgraph and beyond. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2016, Asilomar State Beach/Pacific Grove, CA, USA, July 11-13, 2016*, pages 397–399, 2016.

[30] Anastasia Giachanou and Fabio Crestani. Like it or not : A survey of twitter sentiment analysis methods. *ACM Computing Surveys (CSUR)*, 49(2) :1–41, 2016.

[31] J. Hastad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, FOCS '96, pages 627–. IEEE Computer Society, 1996.

[32] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 :1735–80, 12 1997.

[33] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(2) :13–30, 1963.

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

[35] William H. Inmon. *Building the Data Warehouse*. John Wiley , Sons, Inc., USA, 4rth edition, 2005.

[36] Rajeev Motwani John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Springer Verlag,, 2001.

[37] Sara Keretna, Chee Lim, and Doug Creighton. A hybrid model for named entity recognition using unstructured medical text, 06 2014.

[38] Efthymios Kouloumpis, Theresa Wilson, and Johanna Moore. Twitter sentiment analysis : The good the bad and the omg ! In *Fifth International AAAI conference on weblogs and social media*, 2011.

[39] Omer Levy and Yoav Goldberg. Linguistic regularities in sparse and explicit word representations. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 171–180, June 2014.

[40] Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *CoRR*, abs/1703.03130, 2017.

[41] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999.

[42] Claire Mathieu and Michel de Rougemont. Large very dense subgraphs in a stream of edges. In *ACM-IMS Foundations of Data Science*, pages –, 2020.

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

[44] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[45] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[46] Tomas Mikolov, Scott Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies (NAACL-HLT-2013)*. Association for Computational Linguistics, May 2013.

[47] M. Molloy and B. Reed. A critical point for random graphs with a given degree sequence. *Random Struct. Algorithms*, 6 :161–180, 1995.

[48] 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.

[49] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1) :3–26, January 2007.

[50] Rao Muhammad Adeel Nawab, Mark Stevenson, and Paul Clough. Comparing Medline citations using modified N-grams. *Journal of the American Medical Informatics Association*, 21(1) :105–110, 05 2013.

[51] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove : Global vectors for word representation. In *In EMNLP*, 2014.

[52] P. Ratinaud and S Déjean. Iramuteq : Implémentation de la méthode alceste d'analyse de texte dans un logiciel libre. *HAL*, 2009.

[53] Douglas Rohde, Laura Gonnerman, and David Plaut. An improved model of semantic similarity based on lexical co-occurrence. *Cognitive Science - COGSCI*, 8, 01 2006.

[54] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back propagating errors. cogn. *Nature*, 5, 01 1986.

[55] D.E. Rumelhart and James Mcclelland. On learning the past tenses of english verbs. parallel distributed processing : explorations in the microstructure of cognition. *Psychological and Biological Models*, 2 :216–271, 01 1986.

[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] Sebastian Schuster and Christopher D. Manning. Enhanced english universal dependencies : An improved representation for natural language understanding tasks. In *LREC*, 2016.

[58] Richard Socher, John Bauer, Christopher D. Manning, and Andrew Y. Ng. Parsing with compositional vector grammars. In *In Proceedings of the ACL conference*, 2013.

[59] Richard Socher, Christopher D. Manning, and Andrew Y. Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *In Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*, 2010.

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[61] Guillaume Vimont. Approximation dynamique de clusters dans un graphe social : Méthodes et applications. *Université Panthéon - Assas*, pages 120–122, 2019.

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

[63] Yingce Xia, Tianyu He, Xu Tan, Fei Tian, Di He, and Tao Qin. Tied transformers : Neural machine translation with shared encoder and decoder. In *AAAI*, 2019.

[64] Linzi Xing, Michael J. Paul, and Giuseppe Carenini. Evaluating topic quality with posterior variability. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*, pages 3471–3477. Association for Computational Linguistics, 2019.

[65] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *CoRR*, abs/1708.02709, 2017.

[66] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. Ernie : Enhanced language representation with informative entities, 2019.

Table des figures

Liste des tableaux

A Phrases de la langue Amis

Comme dans le texte, les affixes sont marqués par "-", les infixes par "<...>", les clitiques par "=".

1. Itiya satu a remiad, mi-kasui i lakelal ci Sadaban hananay a tamdaw.
2. Sulinay, adihay=tu k-u ni-pi-kasui.
3. Ta-lumaq=tu cira.
4. Si-pa-qeses cira i liyal.
5. A tubu=tu k-ita.
6. Na sa k-u balucuq n-ira.
7. Pa-tangsul han=tu i liyal.
8. Sulinay, ma-patay k-u balucuq n-ira.
9. Pa-keda han=tu n-ira k-u tireng.
10. U nuka n-u Kawas, si-pa-cahcah cira i niyaruq n-u Balaisan.
11. I cuwa k-ita anini hakira ?
12. Saasaan, ira=tu k-u Balaisan.
13. Lepel han=tu cira.
14. U maan a tamdaw-an k-u ma-tini-ay ?
15. U pulut han-an-ay iri, u panga k-u han.
16. Pa-habay-ay n-umita.
17. Tuduh-en=ita.
18. Tuwa mi-sangaq t-u pa-pulul-an n-i Sadaban.
19. Sa-kapah-en k-u rangat.
20. A pa-habay-an cira n-u Balaisan a niyaruq.
21. Tuwa si-balucuq k-iya pa-habay-ay i ci Sadaban-an.
22. Araw si-balucuq-ay cira.
23. Tayra-an n-ira t-u labi.
24. Sasaan ma-puyapuy cira.
25. Tuwa ka-tangasa-an=tu k-u ka-subuc-an.

26. Sulinay ma-subuc n-ira.
27. U niyaruq hantu n-uhni, ma-banaq t-u ni-ka-subuc n-ira.
28. ka-ta-tengteng han amin k-u panga n-iya wawa.
29. Ma-susu=tu.
30. Ma-dengay=tu.
31. A patay-en k-ina babuy numita.
32. Uruma k-u talaw n-ira t-u a=patay-en.
33. A maan-en=ita ?
34. Tuwa, tumitumian pa-ka-labi sa=tu cira-an.
35. Melaw han n-i Sadaban, u puut.
36. Tanu ulah sa=tu cira.
37. Ira=tu k-iya Balaisan.
38. Mi-kilim i cira-an.
39. Tanu dungudungus sa=tu k-iya Balaisan.
40. U mi-maan-ay k-isu ?
41. Kiya tireng itini ?
42. Tuwa meduk sa=tu tayra i ni-ka-sadak-an n-ira.
43. Pa-sa-tip sa cira.
44. Ira=tu k-iya Balaisan.
45. A ma-lepel=tu k-ita.
46. Tangasa sa cira i ungcung iri.
47. Ci Sadaban hantu iri ma-tini k-u kakawaw numaku.
48. U mi-laliw-ay k-aku t-u Balaisan.
49. U mai-dudu-ay itakuwan k-ira Balaisan.
50. Na ma-lepel k-aku.
51. Urasisa tayni k-aku.
52. Urasisa tangic k-aku.
53. A maan-en=ita.
54. Pa-ta-lumaq-aw-aku k-isu.
55. Kalat-i k-u tangila numaku haw.
56. Han=tu n-ira ci Sadaban.
57. Hay-i sa=tu.
58. Tangasa sa=tu k-uhni.
59. Palita-an n-i Sadaban k-iya isu.
60. Cima k-isu haw aru ?

61. Han=tu n-ira ci Sadaban.
62. Ma-pa-tangasa n-umisu k-aku iri.
63. A caay=tu k-aku ka-pawan t-u ni-pa-urip numisu.
64. ka-si-ayam t-u buhcal-ay.
65. Han=tu n-ira.
66. Sulinay dutuc han=tu nu niyaruq a paysin k-u na-u-suwal n-ira.
67. Ala-en k-u besi numaku.
68. Han=tu n-ira.
69. Sulinay ma-patay sa=tu ci Sadaban.
70. Sa cira.
71. Sa=tu cira.
72. Sa k-aku.
73. Sa cira.
74. Sa=tu k-uheni.
75. Ma-wacay-ay k-uhni.
76. Awaay k-u buduy
77. Ma-'pud=tu k-uhni namaka lutuk haw.
78. Ma-ruqruq itini i 'enar.
79. Mi-awit tu kidudung.
80. Adihay k-u ni-urung t-u kidudung saan iri.
81. Mi-kapet tu sa-sait.
82. Mi-kapet=tu t-u kidudung sa.
83. Mi-sangaq tu talip.
84. Araw ma-banaq k-iya niyaruq itira saan.
85. Ma-banaq tu ngudu.
86. Na caay hen ka-banaq tu ngudu.
87. Dihku=tu k-uhni t-u ka-siqnaw-an iri.
88. Ma-hemek k-uhni.
89. Pa-beli-en k-aku t-u belac iri.
90. Pa-beli-en-aku k-amu t-unian u kidudung.
91. Pa-beli han 'amin n-uhni t-iya taw-an pa-cakay-ay t-u ciyapu.
92. Manay si-buduy sa ku Pangcah.
93. Pa-se-banaq n-iya pa-cakay-ay tu ciyapo sananay.
94. Ma-rimurak ku niyaroq.
95. Mi-cakay t-iya kidudung-an.

Phrases négatives.

1. Caay ka-demec k-u nanum.
2. Caay ka-tala-mana k-u hadui n-ira.
3. Caay=tu ka-wacay iri.

Résumé :

Nous étudions des données de nature diverse sous forme de flux, en particulier :

- Base de données,
- Réseaux sociaux,
- Données de texte.

Pour une base de données qui suit un schéma relationnel, un schéma d'analyse *OLAP* (Online Analytical Processing) définit une des tables de la base de données comme une table d'analyse. Nous supposons que les tuples de la table d'analyse arrivent sous forme d'un flux. Nous étudions l'approximation des requêtes *OLAP*, en échantillonnant de manière non uniforme les tuples du flux sans stocker les données d'analyse et donnons un modèle de préférence dans ce cadre.

Dans le cas du réseau social *Twitter*, nous observons un flux de tweets qui contiennent un tag donné et le transformons en un flux d'arêtes d'un graphe. Nous souhaitons étudier l'existence des grands clusters dans le graphe ainsi obtenu. Nous proposons une méthode d'échantillonnage uniforme qui va associer au graphe un sous-graphe aléatoire et étudions les composantes géantes de ce sous-graphe aléatoire comme témoin des grands clusters du graphe d'origine.

Pour un flux de texte, nous considérons les paires de mots dans une phrase lemmatisée comme des arêtes d'un graphe où les nœuds sont les mots. Nous transformons le flux de texte en flux d'arêtes. Nous échantillonnons les arêtes proportionnellement à la similarité *Word2vec* des mots. Nous analysons ensuite les composantes géantes.

Nous étendons les vecteurs *Word2vec* en prenant en compte la morphologie d'une langue, en particulier la structure des préfixes et des suffixes d'un mot. Les nouveaux vecteurs d'un mot ont 3 composantes : un vecteur pour le préfixe, un vecteur pour la racine et un vecteur pour le suffixe. Nous définissons un vecteur probabiliste pour les phrases.

Sur les trois types de données, nous avons échantillonné selon des distributions précises. Les résultats principaux de cette thèse montrent comment approcher les propriétés de ces données avec ces échantillons.

Descripteurs : Algorithmes de streaming, Approximation, Requêtes OLAP, Préférences, Clustering, Graphes dynamiques, TAL, Morphologie

Abstract :

We study different types of data as a stream :

- Databases,
- Social Networks,
- Text data.

For a database that follows a relational schema, an *OLAP* (Online Analytical Processing) analysis schema defines one of the database tables as an analysis table. We suppose that the tuples of the analysis table arrive as a stream. We study the approximation of *OLAP* queries, by sampling with weights the tuples of the stream without storing the analysis data. We give a *preference* model in this context.

For the social network *Twitter*, we observe a stream of tweets that contain any given tag and transform it into a stream of edges of a graph. We study the existence of large clusters in the generated graph. We propose a uniform sampling method that will associate a random subgraph of the graph and study the giant components of this random subgraph as a witness of the large clusters of the original graph.

For a stream of text, we consider the pairs of words in a lemmatized sentence as edges of a graph where the nodes are the words. We transform the stream of text into a stream of edges. We sample the edges proportionally to the *Word2vec* similarity of the words. We then analyze the giant components.

We extend the classical *Word2vec* vectors, in order to take into account the morphology of the words, i.e. the prefixes, roots and suffixes. A new vector has 3 components, one for the prefix, one for the root and one for the suffix. We define a probabilistic vector associated with the sentences.

With the three types of data, we sampled according to specific distributions. The main results of this thesis show how the properties of these data can be approximated with these samples.

Keywords : Streaming algorithms, Approximation, OLAP queries, Preferences, Clustering, Dynamic graphs,NLP, Morphology

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