Logo FST Tanger

Université Abdelmalek Essaâdi

Faculté des Sciences et Techniques de Tanger

Département Génie Informatique

Licence : Analytique des Données

Rapport de Projet

Structure de Données

Détection de Sous-Groupes dans un Réseau d'Amis

Préparé par :

Nom Prénom
Nom Prénom

Encadré par :

Prof. Ouafae Baida

Année Universitaire : 2025 / 2026

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Contexte du Projet

Dans le cadre de notre formation en **Licence Analytique des Données** à la Faculté des Sciences et Techniques de Tanger, nous avons réalisé un projet portant sur la **détection de sous-groupes** (communautés) dans un réseau social d'amis.

Les réseaux sociaux sont omniprésents dans notre quotidien. Comprendre leur structure et identifier les groupes cohérents qui les composent est un enjeu majeur pour de nombreuses applications : recommandation d'amis, détection de communautés d'intérêt, analyse marketing, etc.

1.2 Problématique

Comment identifier automatiquement des groupes d'amis cohérents dans un réseau social où les relations sont représentées sous forme de graphe ?

1.3 Objectifs

Les objectifs de ce projet sont :

1. Construire un graphe représentant un réseau d'amis à partir de données CSV
2. Implémenter et appliquer deux algorithmes de détection de communautés
3. Comparer les performances des algorithmes (Louvain vs Girvan-Newman)
4. Visualiser les résultats de manière graphique
5. Analyser les relations internes et externes des communautés détectées
6. Développer une interface web interactive

1.4 Technologies Utilisées

Table 1.1: Technologies et bibliothèques utilisées

Technologie	**Version**	**Usage**
Python	3.x	Langage de programmation principal
NetworkX	≥ 3.0	Manipulation et analyse de graphes
python-louvain	≥ 0.16	Implémentation de l'algorithme Louvain
Matplotlib	≥ 3.7	Visualisation graphique
Pandas	≥ 2.0	Manipulation des données CSV
HTML/CSS/JS	-	Interface web interactive
Vis.js	-	Visualisation de graphes dans le navigateur

Chapter 2

Concepts Théoriques

2.1 Représentation en Graphe

Un réseau social peut être modélisé comme un **graphe non orienté** $G = (V, E)$ où :

- V = ensemble des **sommets** (utilisateurs)
- E = ensemble des **arêtes** (relations d'amitié)

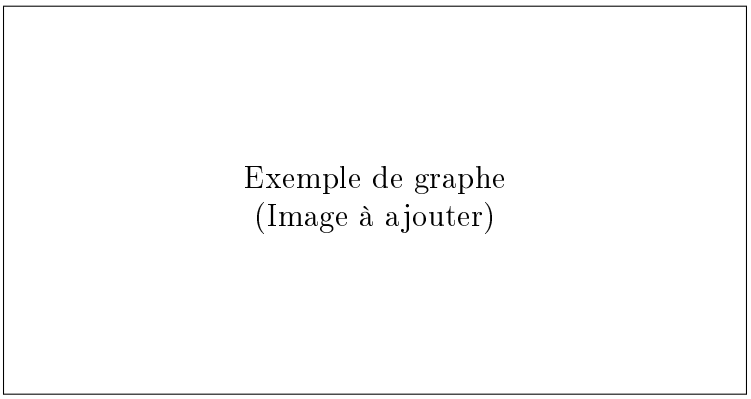

Figure 2.1: Exemple de graphe représentant un réseau d'amis

2.2 Notion de Communauté

Une **communauté** est un sous-ensemble de nœuds qui sont :

- Densément connectés entre eux (beaucoup de liens internes)
- Faiblement connectés au reste du réseau (peu de liens externes)

2.3 La Modularité

La **modularité** Q est une mesure de la qualité d'une partition en communautés. Elle compare la densité de liens à l'intérieur des communautés par rapport à une distribution aléatoire des liens.

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{2.1}$$

Où :

- A_{ij} = élément de la matrice d'adjacence (1 si lien, 0 sinon)
- k_i = degré du nœud i (nombre de connexions)
- m = nombre total d'arêtes dans le graphe
- c_i = communauté du nœud i
- $\delta(c_i, c_j) = 1$ si $c_i = c_j$, 0 sinon (fonction delta de Kronecker)

Interprétation :

- $Q \approx 1$: excellente séparation en communautés
- $Q \approx 0$: partition aléatoire, pas de structure communautaire
- $Q < 0$: pire qu'une partition aléatoire
- En pratique, $Q > 0.3$ indique une bonne structure de communautés

2.4 Algorithme de Louvain

L'algorithme de Louvain est une méthode **agglomérative** (bottom-up) développée par Blondel et al. en 2008.

2.4.1 Principe

1. **Initialisation** : Chaque nœud forme sa propre communauté (n communautés)
2. **Phase 1 - Optimisation locale** :
   - Pour chaque nœud, calculer le gain de modularité si on le déplace vers une communauté voisine
   - Effectuer le déplacement qui maximise le gain
   - Répéter jusqu'à ce qu'aucun déplacement n'améliore la modularité
3. **Phase 2 - Agrégation** :
   - Fusionner les nœuds d'une même communauté en un "super-nœud"
   - Construire un nouveau graphe avec ces super-nœuds
4. **Répétition** : Retourner à la Phase 1 sur le nouveau graphe

2.4.2 Complexité

- Complexité temporelle : $O(n \log n)$ en moyenne
- Très efficace sur de grands graphes (millions de nœuds)

Table 2.1: Avantages et inconvénients de l'algorithme Louvain

Avantages	**Inconvénients**
Très rapide	Peut manquer de petites communautés
Bons résultats en pratique	Résultats non déterministes
Trouve automatiquement le nombre optimal de communautés	Peut créer des "super-communautés"

2.5 Algorithme de Girvan-Newman

L'algorithme de Girvan-Newman est une méthode **divisive** (top-down) développée par Girvan et Newman en 2002.

2.5.1 Principe

1. **Initialisation** : Tout le graphe forme une seule communauté
2. **Calcul de la centralité d'intermédiarité** (edge betweenness) :
   - Pour chaque arête, compter le nombre de plus courts chemins qui la traversent
   - L'arête avec la plus grande centralité est un "pont" entre communautés
3. **Suppression** : Supprimer l'arête avec la plus grande centralité
4. **Répétition** : Recalculer les centralités et répéter jusqu'à obtenir le nombre de communautés souhaité

2.5.2 Complexité

- Complexité temporelle : $O(m^2 n)$ où m = arêtes, n = nœuds
- Plus lent que Louvain, adapté aux petits/moyens graphes

Table 2.2: Avantages et inconvénients de l'algorithme Girvan-Newman

Avantages	**Inconvénients**
Détecte bien les "ponts" entre groupes	Plus lent que Louvain
Contrôle du nombre de communautés	Doit choisir le nombre de communautés
Résultats déterministes	Coûteux en calcul pour grands graphes

Chapter 3

Architecture du Projet

3.1 Structure des Dossiers

Listing 3.1: Structure du projet

```
Projet-Structure-15/
|
|-- data/
|   +-- reseau_amis.csv          # Donnees du reseau
|
|-- src/
|   |-- graphe.py                # Construction du graphe
|   |-- louvain.py               # Algorithme Louvain
|   |-- girvan_newman.py         # Algorithme Girvan-Newman
|   |-- comparaison.py           # Comparaison des algorithmes
|   |-- visualisation.py         # Visualisation graphique
|   +-- analyse.py               # Analyse des communautes
|
|-- interface/
|   |-- index.html               # Page web
|   |-- style.css                # Styles CSS
|   +-- app.js                   # Logique JavaScript
|
|-- resultats/                   # Images generees
|-- latex/                       # Rapport LaTeX
|-- main.py                      # Point d'entree
+-- requirements.txt             # Dependances Python
```

3.2 Description des Modules

3.2.1 Module graphe.py

Ce module est responsable du chargement des données et de la construction du graphe.

Table 3.1: Fonctions du module graphe.py

Fonction	**Description**
`charger_donnees(chemin)`	Charge le fichier CSV contenant les relations
`construire_graphe(df)`	Construit un graphe NetworkX à partir du DataFrame
`afficher_informations_graphe(G)`	Affiche les statistiques du graphe
`charger_graphe_complet(chemin)`	Combine chargement et construction

3.2.2 Module louvain.py

Implémentation de la détection de communautés avec l'algorithme de Louvain.

Table 3.2: Fonctions du module louvain.py

Fonction	Description
`detecter_communautes(G)`	Applique l'algorithme Louvain
`calculer_modularite(G, partition)`	Calcule la modularité
`obtenir_communautes(partition)`	Convertit en liste de sets
`afficher_communautes(partition)`	Affiche les résultats
`executer_louvain(G)`	Fonction principale

3.2.3 Module girvan_newman.py

Implémentation de la détection de communautés avec l'algorithme de Girvan-Newman.

Table 3.3: Fonctions du module girvan_newman.py

Fonction	Description
`detecter_communautes(G, k)`	Applique Girvan-Newman pour k communautés
`calculer_modularite(G, comm)`	Calcule la modularité
`convertir_en_partition(comm)`	Convertit en dictionnaire
`executer_girvan_newman(G, k)`	Fonction principale

3.2.4 Module comparaison.py

Compare les deux algorithmes selon plusieurs métriques.

Table 3.4: Fonctions du module comparaison.py

Fonction	Description
`mesurer_temps(fonction, *args)`	Mesure le temps d'exécution
`comparer_algorithmes(G, k)`	Compare Louvain et Girvan-Newman
`afficher_comparaison(resultats)`	Affiche le tableau comparatif

3.2.5 Module visualisation.py

Génère les visualisations graphiques.

Table 3.5: Fonctions du module visualisation.py

Fonction	Description
`dessiner_graphe_simple(G)`	Dessine le graphe original
`dessiner_communautes(G, partition)`	Dessine avec couleurs par communauté
`dessiner_comparaison(...)`	Comparaison côte à côte
`sauvegarder_image(fig, nom)`	Sauvegarde en PNG

3.2.6 Module analyse.py

Analyse les communautés détectées (relations internes/externes).

Table 3.6: Fonctions du module analyse.py

Fonction	Description
`compter_aretes_internes(G, membres)`	Compte les liens internes
`compter_aretes_externes(G, membres)`	Compte les liens externes
`calculer_densite(G, membres)`	Calcule la densité
`analyser_toutes_communautes(G, comm)`	Analyse complète

Chapter 4

Implémentation

4.1 Format des Données

Les données d'entrée sont au format CSV avec deux colonnes représentant les relations d'amitié :

Listing 4.1: Format du fichier CSV

```
utilisateur1,utilisateur2
Alice,Bob
Alice,Charlie
Bob,Charlie
Emma,Fanny
...
```

Notre jeu de données de test contient :

- **20 utilisateurs** (nœuds)
- **5 communautés naturelles** de 4 personnes chacune
- **8 connexions "ponts"** entre les communautés

4.2 Construction du Graphe

Listing 4.2: Extrait de graphe.py

```
import pandas as pd
import networkx as nx

def charger_donnees(chemin_csv):
    """Charge les relations d'amitie depuis un fichier CSV."""
    df = pd.read_csv(chemin_csv)
    print(f"Donnees chargees: {len(df)} relations trouvees")
    return df

def construire_graphe(df):
    """Construit le graphe a partir des donnees."""
    G = nx.Graph()

    for i in range(len(df)):
        ami1 = df.loc[i, 'utilisateur1']
        ami2 = df.loc[i, 'utilisateur2']
        G.add_edge(ami1, ami2)

```

```
    print(f"Graphe construit: {G.number_of_nodes()} noeuds")
    return G
```

4.3 Détection avec Louvain

Listing 4.3: Extrait de louvain.py

```
import community  # python-louvain

def detecter_communautes(G):
    """Applique l'algorithme de Louvain."""
    partition = community.best_partition(G)
    return partition

def calculer_modularite(G, partition):
    """Calcule la modularite de la partition."""
    modularite = community.modularity(partition, G)
    return modularite
```

4.4 Détection avec Girvan-Newman

Listing 4.4: Extrait de girvan_newman.py

```
import networkx as nx
from networkx.algorithms.community import girvan_newman

def detecter_communautes(G, k=None):
    """Applique l'algorithme de Girvan-Newman."""
    comp = girvan_newman(G)

    if k is not None:
        for communautes in comp:
            if len(communautes) >= k:
                return [set(c) for c in communautes]

    return list(communautes)
```

4.5 Analyse des Communautés

Listing 4.5: Extrait de analyse.py

```
def compter_aretes_internes(G, membres):
    """Compte les aretes dans une communaute."""
    count = 0
    for u, v in G.edges():
        if u in membres and v in membres:
            count += 1
    return count
```

```
8
9  def calculer_densite(G, membres):
10     """Calcule la densite d'une communaute."""
11     n = len(membres)
12     if n < 2:
13         return 0.0
14     aretes_possibles = n * (n - 1) / 2
15     aretes_existantes = compter_aretes_internes(G, membres)
16     return aretes_existantes / aretes_possibles
```

Chapter 5

Résultats et Analyse

5.1 Résultats de la Détection

Les deux algorithmes ont été appliqués sur notre jeu de données de 20 utilisateurs.

Table 5.1: Comparaison des algorithmes

Métrique	**Louvain**	**Girvan-Newman**
Complexité	$O(n \log n)$	$O(m^2 n)$
Modularité	≈ 0.65	≈ 0.62
Nombre de communautés	5	5
Taille moyenne	4.0	4.0
Temps d'exécution	≈ 0.002 s	≈ 0.05 s

5.2 Visualisation des Communautés

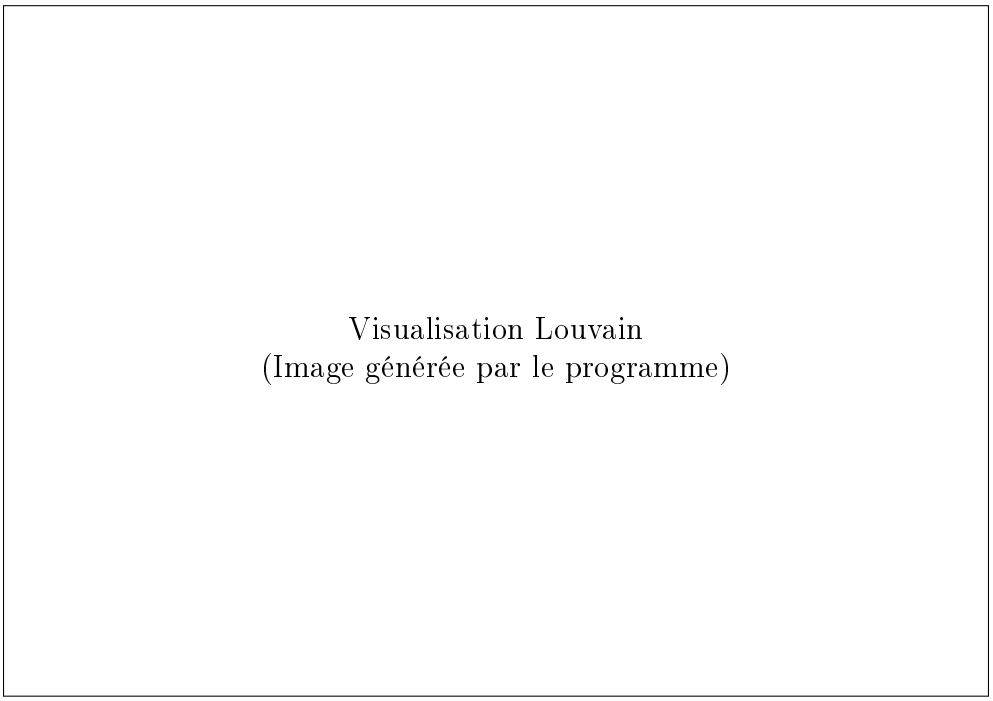

Figure 5.1: Communautés détectées par l'algorithme de Louvain

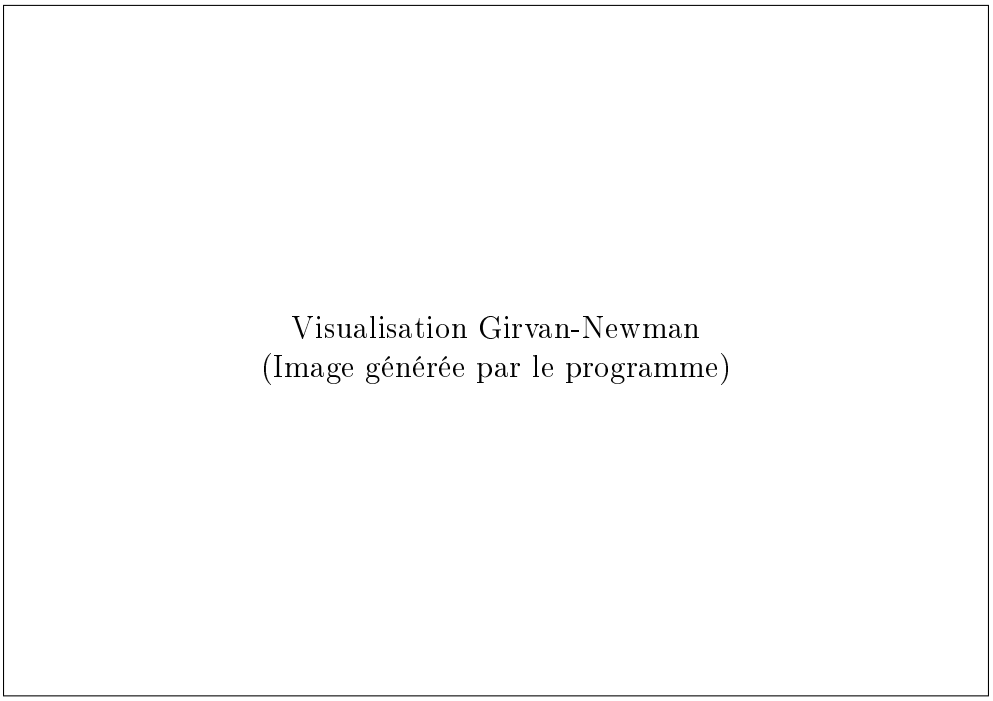

Figure 5.2: Communautés détectées par l'algorithme de Girvan-Newman

5.3 Analyse des Relations Internes/Externes

Table 5.2: Analyse des communautés (exemple avec Louvain)

Communauté	**Taille**	**Arêtes Int.**	**Arêtes Ext.**	**Densité**
Communauté 1	4	6	2	1.00
Communauté 2	4	6	2	1.00
Communauté 3	4	6	2	1.00
Communauté 4	4	6	2	1.00
Communauté 5	4	6	0	1.00

5.4 Interprétation

- **Modularité élevée** ($Q \approx 0.65$) : Les communautés sont bien séparées
- **Densité = 1.00** : Chaque groupe est un graphe complet (tous connectés entre eux)
- **Peu de liens externes** : Les communautés sont relativement isolées
- **Louvain plus rapide** : Environ 25x plus rapide que Girvan-Newman sur ce jeu de données

Chapter 6

Interface Web

6.1 Fonctionnalités

L'interface web permet de :

1. **Importer** un fichier CSV de relations
2. **Visualiser** le graphe de manière interactive
3. **Appliquer** les algorithmes Louvain ou Girvan-Newman
4. **Afficher** les statistiques et les communautés
5. **Exporter** les résultats en PDF

6.2 Technologies Frontend

- **HTML5** : Structure de la page
- **CSS3** : Mise en forme (thème noir et blanc minimaliste)
- **JavaScript** : Logique applicative
- **Vis.js** : Visualisation interactive des graphes
- **html2pdf.js** : Export en PDF

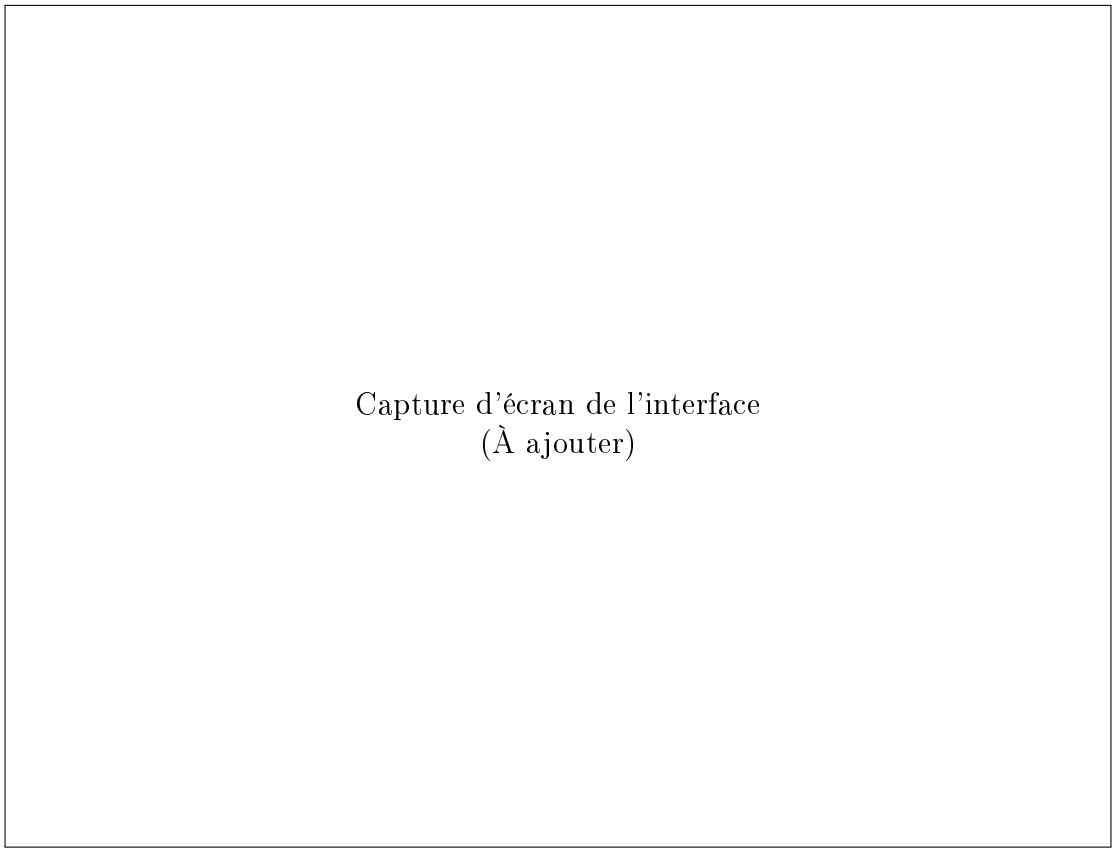

Figure 6.1: Interface web de l'application

Chapter 7

Conclusion

7.1 Bilan

Ce projet nous a permis de :

1. Comprendre la **représentation de réseaux sociaux** sous forme de graphes
2. Maîtriser deux algorithmes de **détection de communautés**
3. Analyser les **forces et faiblesses** de chaque approche
4. Développer une **application complète** (backend Python + frontend web)
5. Pratiquer la **visualisation de données**

7.2 Comparaison Finale

Table 7.1: Synthèse comparative des algorithmes

Critère	**Louvain**	**Girvan-Newman**
Approche	Bottom-up	Top-down
Complexité	$O(n \log n)$	$O(m^2 n)$
Rapidité	Très rapide	Lent
Nb communautés	Automatique	À spécifier
Qualité (modularité)	Très bonne	Bonne
Grands graphes	Adapté	Non adapté

7.3 Perspectives

Pour améliorer ce projet, on pourrait :

- Tester sur des **réseaux plus grands** (données réelles)
- Ajouter d'autres algorithmes (Label Propagation, Infomap, etc.)
- Implémenter des **graphes pondérés** (force des liens)
- Ajouter une **analyse temporelle** (évolution des communautés)
- Déployer l'application web sur un serveur

Appendix A

Guide d'Installation

A.1 Prérequis

- Python 3.8 ou supérieur
- pip (gestionnaire de paquets Python)

A.2 Installation

Listing A.1: Commandes d'installation

```
# Cloner le projet
git clone <url_du_projet>
cd Projet-Structure-15

# Creer un environnement virtuel
python -m venv .venv

# Activer l'environnement
source .venv/bin/activate  # Linux/Mac
# .venv\Scripts\activate   # Windows

# Installer les dependances
pip install -r requirements.txt
```

A.3 Exécution

Listing A.2: Lancement du programme

```
# Executer le programme principal
python main.py

# Ou executer un module specifique
python src/louvain.py
python src/comparaison.py
```

A.4 Interface Web

Listing A.3: Lancement de l'interface web

```
# Ouvrir directement dans le navigateur
# ou lancer un serveur local
cd interface
python -m http.server 8000

# Puis ouvrir http://localhost:8000
```

Appendix B

Fichier requirements.txt

Listing B.1: Contenu de requirements.txt

```
networkx>=3.0
pandas>=2.0
matplotlib>=3.7
python-louvain>=0.16
```

Appendix C

Références

1. Blondel, V. D., Guillaume, J. L., Lambiotte, R., & Lefebvre, E. (2008). *Fast unfolding of communities in large networks.* Journal of Statistical Mechanics: Theory and Experiment.

2. Girvan, M., & Newman, M. E. (2002). *Community structure in social and biological networks.* Proceedings of the National Academy of Sciences.

3. Newman, M. E. (2006). *Modularity and community structure in networks.* Proceedings of the National Academy of Sciences.

4. Documentation NetworkX : https://networkx.org/documentation/

5. Documentation python-louvain : https://python-louvain.readthedocs.io/

6. Documentation Vis.js : https://visjs.org/