Documentation du Projet

Détection de Communautés dans un Réseau d'Amis

1 Présentation du Projet

1.1 Objectif Principal

Identifier des **sous-groupes cohérents** (communautés) dans un réseau social de 20-30 utilisateurs. Une communauté est un groupe d'utilisateurs plus connectés entre eux qu'avec le reste du réseau.

1.2 Ce que nous allons faire

1. Créer un fichier de données avec les relations d'amitié (CSV)
2. Construire un graphe à partir de ces données
3. Appliquer 2 algorithmes de détection de communautés
4. Visualiser les résultats graphiquement
5. Comparer les algorithmes
6. Analyser les relations internes et externes

2 Structure des Dossiers

```
Projet-Structure-15/
|
|-- data/
|   +-- reseau_amis.csv       # Les donnees (relations d'amitie)
|
|-- src/
|   |-- graphe.py             # Construire le graphe
|   |-- louvain.py            # Algorithme 1 : Louvain
|   |-- girvan_newman.py      # Algorithme 2 : Girvan-Newman
|   |-- comparaison.py        # Comparer les 2 algorithmes
|   |-- visualisation.py      # Dessiner les graphes
|   +-- analyse.py            # Analyser les communautes
|
|-- resultats/                # Images generees
|-- latex/                    # Ce document
|-- main.py                   # Programme principal
+-- requirements.txt          # Bibliotheques a installer
```

3 Description de Chaque Fichier

3.1 data/reseau_amis.csv

Rôle : Contient les données du réseau social.

Format :

```
utilisateur1,utilisateur2
Alice,Bob
Alice,Charlie
Bob,Charlie
```

Contenu actuel :

- 20 utilisateurs (nœuds)
- 5 groupes naturels de 4 personnes chacun
- Des connexions entre les groupes (ponts)

3.2 src/graphe.py

Rôle : Charger les données et construire le graphe.

Fonctions :

- `charger_donnees(chemin)` – Lit le fichier CSV
- `construire_graphe(df)` – Crée le graphe NetworkX
- `afficher_informations_graphe(G)` – Affiche les stats (nœuds, arêtes, densité...)
- `charger_graphe_complet(chemin)` – Fait tout en une seule étape

Bibliothèques utilisées : pandas, networkx

3.3 src/louvain.py

Rôle : Détecter les communautés avec l'algorithme de Louvain.

Fonctions prévues :

- `detecter_communautes(G)` – Applique Louvain et retourne les groupes
- `calculer_modularite(G, partition)` – Calcule la qualité de la partition
- `afficher_communautes(partition)` – Affiche les membres de chaque groupe

Bibliothèque utilisée : python-louvain (community)

3.4 src/girvan_newman.py

Rôle : Détecter les communautés avec l'algorithme de Girvan-Newman.

Fonctions prévues :

- `detecter_communautes(G, k)` – Applique Girvan-Newman pour k communautés
- `calculer_modularite(G, communautes)` – Calcule la qualité
- `afficher_communautes(communautes)` – Affiche les groupes

Bibliothèque utilisée : networkx (intégré)

3.5 src/comparaison.py

Rôle : Comparer les résultats des deux algorithmes.

Fonctions prévues :

- `comparer_algorithmes(G)` – Exécute les 2 algos et compare
- `afficher_tableau_comparaison(resultats)` – Affiche un tableau récapitulatif
- `mesurer_temps_execution(algo, G)` – Mesure le temps de chaque algo

Métriques comparées :

- Modularité (qualité de la partition)
- Temps d'exécution
- Nombre de communautés détectées
- Taille moyenne des communautés

3.6 src/visualisation.py

Rôle : Créer les visualisations graphiques.

Fonctions prévues :

- `dessiner_graphe_simple(G)` – Dessine le graphe sans couleurs
- `dessiner_communautes(G, partition, titre)` – Dessine avec couleurs par communauté
- `dessiner_comparaison(G, partition1, partition2)` – Côte à côte
- `sauvegarder_image(fig, nom)` – Sauvegarde dans resultats/

Bibliothèque utilisée : matplotlib

3.7 src/analyse.py

Rôle : Analyser les communautés détectées.

Fonctions prévues :

- `compter_aretes_internes(G, communaute)` – Arêtes dans le groupe
- `compter_aretes_externes(G, communaute)` – Arêtes vers d'autres groupes
- `calculer_densite(G, communaute)` – Densité du groupe
- `analyser_toutes_communautes(G, partition)` – Analyse complète
- `afficher_rapport(analyse)` – Affiche les résultats

Métriques calculées :

- Arêtes internes vs externes
- Ratio interne/externe
- Densité de chaque communauté

3.8 main.py

Rôle : Point d'entrée qui exécute tout le programme.

Étapes :

1. Charger le graphe depuis le CSV
2. Afficher les informations du graphe
3. Appliquer Louvain
4. Appliquer Girvan-Newman
5. Comparer les deux algorithmes
6. Visualiser les résultats
7. Analyser les communautés
8. Sauvegarder les images

4 Les Algorithmes

4.1 Algorithme de Louvain

Approche : Bottom-up (agglomérative)

Principe :

1. Au début : chaque personne = sa propre communauté
2. On essaie de déplacer chaque personne vers une communauté voisine

3. Si ça améliore la modularité, on fait le déplacement
4. On répète jusqu'à ce qu'on ne puisse plus améliorer
5. On fusionne les communautés en "super-nœuds" et on recommence

Avantages :

- Très rapide
- Bons résultats
- Trouve automatiquement le nombre de communautés

Inconvénients :

- Peut manquer de petites communautés

4.2 Algorithme de Girvan-Newman

Approche : Top-down (divisive)

Principe :

1. Au début : tout le monde dans une seule grande communauté
2. On calcule l'importance de chaque arête (betweenness)
3. On supprime l'arête la plus importante (le "pont")
4. On répète jusqu'à avoir le nombre de communautés voulu

Avantages :

- Détecte bien les ponts entre groupes
- On contrôle le nombre de communautés

Inconvénients :

- Plus lent que Louvain
- Il faut choisir le nombre de communautés

5 La Modularité

La **modularité** mesure la qualité d'une partition en communautés.

Formule :

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

Interprétation :

- Q proche de 1 = excellente séparation en communautés
- Q proche de 0 = partition aléatoire
- Q négatif = pire qu'aléatoire

En pratique, une modularité > 0.3 indique une bonne structure de communautés.

6 Analyse des Relations

6.1 Relations Internes

Connexions entre membres de la **même** communauté.

$$\text{Arêtes internes} = \text{connexions dans le groupe}$$

6.2 Relations Externes

Connexions entre membres de communautés **différentes**.

$$\text{Arêtes externes} = \text{connexions vers d'autres groupes}$$

6.3 Densité Interne

Mesure à quel point un groupe est connecté.

$$\text{Densité} = \frac{\text{arêtes existantes}}{\text{arêtes possibles}} = \frac{2 \times \text{arêtes}}{n \times (n-1)}$$

Une bonne communauté a :

- Beaucoup d'arêtes internes
- Peu d'arêtes externes
- Densité interne élevée

7 Comparaison Prévue

Critère	**Louvain**	**Girvan-Newman**
Approche	Bottom-up	Top-down
Vitesse	Rapide	Lent
Nb communautés	Automatique	À choisir
Qualité	Très bonne	Bonne

8 Comment Exécuter le Projet

8.1 Installation

```
# Creer un environnement virtuel
python -m venv .venv
source .venv/bin/activate

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

8.2 Exécution

```
python main.py
```

8.3 Dépendances (requirements.txt)

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

9 Résultats Attendus

Le programme va générer :

1. **Affichage console :**
   - Informations du graphe (nœuds, arêtes, densité)
   - Communautés détectées par chaque algorithme
   - Tableau de comparaison
   - Analyse des relations internes/externes
2. **Images dans resultats/ :**
   - graphe_original.png – Le réseau sans couleurs
   - graphe_louvain.png – Communautés Louvain
   - graphe_girvan_newman.png – Communautés Girvan-Newman
   - comparaison.png – Les deux côte à côte