

UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À

L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE APPLIQUÉES

PAR

GAICIRE FALL

UNE APPROCHE HYBRIDE POUR LA DÉTECTION DES COMMUNAUTÉS DANS LES
GRAPHES

JUIN 2025

Université du Québec à Trois-Rivières

Service de la bibliothèque

Avertissement

L'auteur de ce mémoire, de cette thèse ou de cet essai a autorisé l'Université du Québec à Trois-Rivières à diffuser, à des fins non lucratives, une copie de son mémoire, de sa thèse ou de son essai.

Cette diffusion n'entraîne pas une renonciation de la part de l'auteur à ses droits de propriété intellectuelle, incluant le droit d'auteur, sur ce mémoire, cette thèse ou cet essai. Notamment, la reproduction ou la publication de la totalité ou d'une partie importante de ce mémoire, de cette thèse et de son essai requiert son autorisation.

RESUMÉ

De nombreuses recherches ont exploré la détection de communautés dans les réseaux depuis sa découverte il y a plusieurs années, aboutissant au développement de divers algorithmes et approches pour résoudre ce problème complexe. Ces méthodes incluent la classification, l'optimisation de fonctions objectifs et les concepts issus de la théorie des graphes. Néanmoins, la définition précise de la détection communautaire demeure encore floue.

Dans cette étude, nous proposons une nouvelle approche hybride de détection, combinant l'algorithme de Louvain et la classification spectrale. Cette approche a été initiée par les chercheurs Zhang et Newman. L'algorithme hybride tire parti de la rapidité de détection des communautés offerte par l'algorithme de Louvain, tout en bénéficiant de la précision dans la séparation des clusters grâce aux propriétés spectrales du graphe fournies par la classification spectrale. L'objectif de cette approche est d'optimiser la précision dans l'identification et la caractérisation des structures communautaires complexes présentes dans divers types de réseaux interconnectés.

Notre recherche se concentrera sur l'implémentation de cet algorithme hybride afin de comparer les résultats obtenus avec ceux issus des méthodes basées sur des techniques d'optimisation, telles que l'algorithme de Louvain, et ceux issus de la classification spectrale développée par Mazouz.

En résumé, notre travail vise à étendre les recherches antérieures en adoptant une approche hybride pour vérifier si la combinaison de ces deux algorithmes améliore l'efficacité de la détection et pour comparer les résultats obtenus avec ceux des deux méthodes individuelles.

ABSTRACT

Many studies have explored community detection in networks since its discovery several years ago, leading to the development of various algorithms and approaches to tackle this complex problem. These methods include classification, optimization of objective functions, and concepts from graph theory. However, the precise definition of community detection remains somewhat elusive.

In this study, we propose a new hybrid detection approach that combines the Louvain algorithm with spectral classification. This approach was initiated by researchers Zhang and Newman. The hybrid algorithm leverages the rapid community detection offered by the Louvain algorithm while benefiting from the precision in cluster separation provided by the spectral properties of the graph through spectral classification. The goal of this hybrid approach is to enhance accuracy in identifying and characterizing complex community structures present in various types of interconnected networks.

Our research will focus on implementing this hybrid algorithm to compare the results with those obtained from optimization-based methods, such as the Louvain algorithm, as well as with those from spectral classification methods developed by Mazzouz. In summary, our work aims to extend previous studies by adopting a hybrid approach to determine whether combining these two algorithms improves detection efficiency and to compare the results with those from the two individual methods.

REMERCIEMENTS

Je commence par exprimer ma gratitude envers Dieu pour sa grâce et sa guidance tout au long de la rédaction de mon mémoire.

Je souhaite également exprimer ma reconnaissance à différentes personnes qui ont joué un rôle crucial dans cette réussite :

Tout d'abord, à mon père, dont les sacrifices et le soutien inconditionnel ont été une source d'inspiration constante. Sa présence et son soutien moral m'ont encouragé à persévérer dans les moments difficiles.

À ma mère, dont l'amour et les encouragements m'ont toujours poussée à aller de l'avant. Sa sagesse et son soutien inébranlable ont été des piliers tout au long de ce parcours.

À mes sœurs, dont le soutien constant et les encouragements ont été précieux tout au long de ce parcours. Leur présence et leur écoute attentive ont été un soutien réconfortant.

À mon mari, dont le soutien indéfectible et les encouragements sans faille ont été une source de force et de motivation. Son soutien logistique et émotionnel m'ont permis de me concentrer pleinement sur mes recherches.

Enfin, à mon directeur de recherche, dont les conseils éclairés, l'expertise et le soutien précieux ont grandement enrichi mon travail. Ses orientations avisées ont été essentielles pour naviguer à travers les défis académiques et pour atteindre l'excellence dans ce projet.

La contribution de chacune de ces personnes a été déterminante dans la réussite de ce projet académique, et je leur suis profondément reconnaissante pour leur soutien inestimable.

Table des matières

1 INTRODUCTION.....	10
1.1 Cadre de référence	10
1.1 Problématique.....	13
1.2 Apport du mémoire	15
1.3 Agencement du mémoire	16
2 ETAT DE L'ART.....	17
2.1 Postulat de base	17
2.1.1 Graphe non orienté.....	18
2.1.2 Graphe orienté	21
2.2 Analyse spectrale de graphes	23
2.2.1 Définition	23
2.2.2 Exemple concret	24
2.2.3 Application d'une méthode de classification	26
2.3 Notions de communauté	27
2.4 Méthode de détection de communautés.....	32
2.4.1 Approche d'optimisation de la Modularité	32
2.4.2 Approche du chemin aléatoire.....	35
2.4.3 Méthode de classification spectrale	36
2.5 Conclusion.....	38
3 MÉTHODE PROPOSÉE : APPROCHE HYBRIDE.....	39
3.1 Présentation	39

3.2	Principe de la méthode hybride	39
3.3	Fonctionnement	40
3.3.1	Détection initiale des communautés avec Louvain	43
3.3.2	Construction de la matrice Laplacienne.....	47
3.3.3	Classification spectrale sur la matrice Laplacienne	49
3.3.4	Clustering spectral à l'aide de l'algorithme k-moyennes.....	54
3.4	Visualisation des communautés détectées par l'algorithme Hybride	57
3.5	Description algorithmique.....	58
3.6	Conclusion.....	59
4	IMPLÉMENTATION ET EXPÉRIMENTATION	60
4.1	Implémentation de l'Algorithme	60
4.1.1	Environnement.....	61
4.1.2	Package	62
4.2	Mise en œuvre de l'algorithme.....	64
4.2.1	Chargement des Bibliothèques	64
4.2.2	Fonctions Principales	64
4.3	Exécution de l'algorithme.....	77
	Étape 1 : Génération du Graphe	77
	Étape 2 : Visualisation Initiale	77
	Étape 3 : Calcul de la Matrice Laplacienne	77
	Étape 4 : Détection des Communautés	77
	Étape 5 : Classification Spectrale.....	78
	Étape 6 : Visualisation des Communautés.....	78
	Pseudo Code :	79
4.4	Évaluation Comparative de la modularité sous variation du paramètre de mixage	80
4.4.1	Paramètre de mixage.....	80
4.4.2	Notion de modularité	81
□	Test pour n=100.....	82

□	Test pour $n=1000$	84
□	Test pour $n=10000$	86
4.5	Visualisation des différents graphes finaux	89
4.6	Conclusion.....	91
5	CONCLUSION GÉNÉRALE ET PERSPECTIVES	92
5.1	Conclusion générale.....	92
5.2	Perspectives	93

Table des figures

Figure 1: Jeu représenté par la théorie des graphes	10
Figure 2: Graphes avec plusieurs communautés.....	11
Figure 3: Détection de quatre communautés dans un réseau de graphe.....	12
Figure 4: Graphe simple	18
Figure 5: Graphe connexe et non-connexe	19
Figure 6: Graphe complet et Graphe non complet	19
Figure 7: Graphe biparti	20
Figure 8: Graphe avec sa matrice d'adjacence	20
Figure 9: Graphe orienté.....	21
Figure 10: Graphe orienté avec sa matrice d'adjacence.....	22
Figure 11: Exemple de graphe avec trois communautés disjointes	30
Figure 12: Exemple de communautés chevauchantes.....	31
Figure 13: Exemple de Communautés fortes et faibles	32
Figure 14: Graphe G avec sa matrice d'adjacence (A)	43
Figure 15: Résultats des vecteurs propres dans R.....	52
Figure 16: Résultat finale de l'algorithme.....	57
Figure 17: Variation de la modularité en fonction du paramètre de mixage μ pour le graphe de 100 nœuds.....	83
Figure 18: Variation de la modularité en fonction du paramètre de mixage	85
Figure 19: Variation de la modularité en fonction du paramètre de mixage μ pour le graphe de 10000 nœuds.....	87
Figure 20: Graphe final détecté par l'algorithme Louvain	89
Figure 21: Graphe final détecté par l'algorithme Mazzouz.....	90

Liste des tableaux

Tableau 1: Avantages et limites de l'approche d'optimisation.....	34
Tableau 2: Avantages et Limites de l'approche du chemin aléatoire	36
Tableau 3: Avantages et Limites de l'approche de la classification spectrale	38
Tableau 4: Vecteurs propres des 3 plus petites valeurs propres	53
Tableau 5: Résultats de la modularité pour le graphe de 100 nœuds en variant le paramètre de mixage μ	82
Tableau 6: Résultats de la modularité pour le graphe de 1000 nœuds en variant le paramètre de mixage μ	84
Tableau 7: Résultats de la modularité pour le graphe de 10000 nœuds en variant le paramètre de mixage μ	86

1 INTRODUCTION

1.1 Cadre de référence

La théorie des graphes est une discipline mathématique et informatique qui étudie les graphes, qui sont des représentations abstraites des connexions entre des objets. Elle est utilisée dans plusieurs domaines. On peut utiliser la théorie des graphes pour aider à étudier plusieurs domaines comme la biologie, l'économie, et bien plus encore [4]. La résolution et la compréhension des problèmes complexes sont facilitées par la représentation en graphiques [1]. Dans le domaine de l'Internet comme il est possible de l'observer sur la figure 1, les sites web sont des sommets et les hyperliens entre eux sont des arêtes, ce qui facilite l'analyse de la structure du web [36]. Dans le domaine du transport, les stations de transport peuvent être visualisées comme des sommets et les itinéraires comme des arêtes dans un graphe [43]. De même, en électricité, les éléments d'un circuit électrique peuvent être envisagés comme des sommets et leurs connexions comme des arêtes dans un graphe.

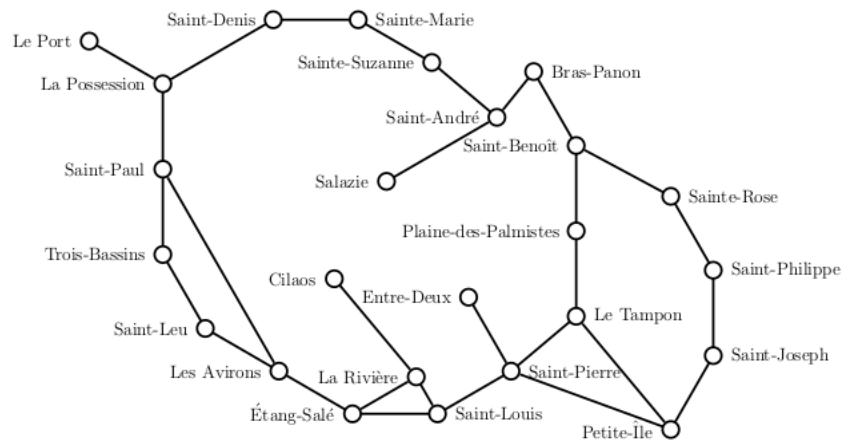


Figure 1: Jeu représenté par la théorie des graphes

De nombreux chercheurs se sont naturellement intéressés à ce type de graphes, et récemment, ils ont introduit un nouveau concept : la recherche de communautés dans les graphes [19]. Dans le domaine des réseaux, une communauté se définit comme un groupe de nœuds qui sont particulièrement reliés entre eux et faiblement reliés au reste du réseau [17]. Il peut s'agir, par exemple, d'individus qui échangent beaucoup entre eux et peu avec les autres [27]. En biologie, une communauté représente un système au sein duquel des organismes vivants partagent un environnement commun et interagissent, comme dans les réseaux d'interactions biologiques [8]. La notion de communauté est également un concept du droit qui désigne un groupe de personnes possédant et jouissant, de façon indivise, d'un patrimoine en commun. Au sens des réseaux sociaux, une communauté est formée par un ensemble d'individus qui interagissent. En d'autres termes, une communauté représente un ensemble de nœuds présentant une forte cohésion interne en termes de connexions ou de similarités, mais pouvant être moins liés à d'autres parties du réseau [22]

(Voir figure 2).

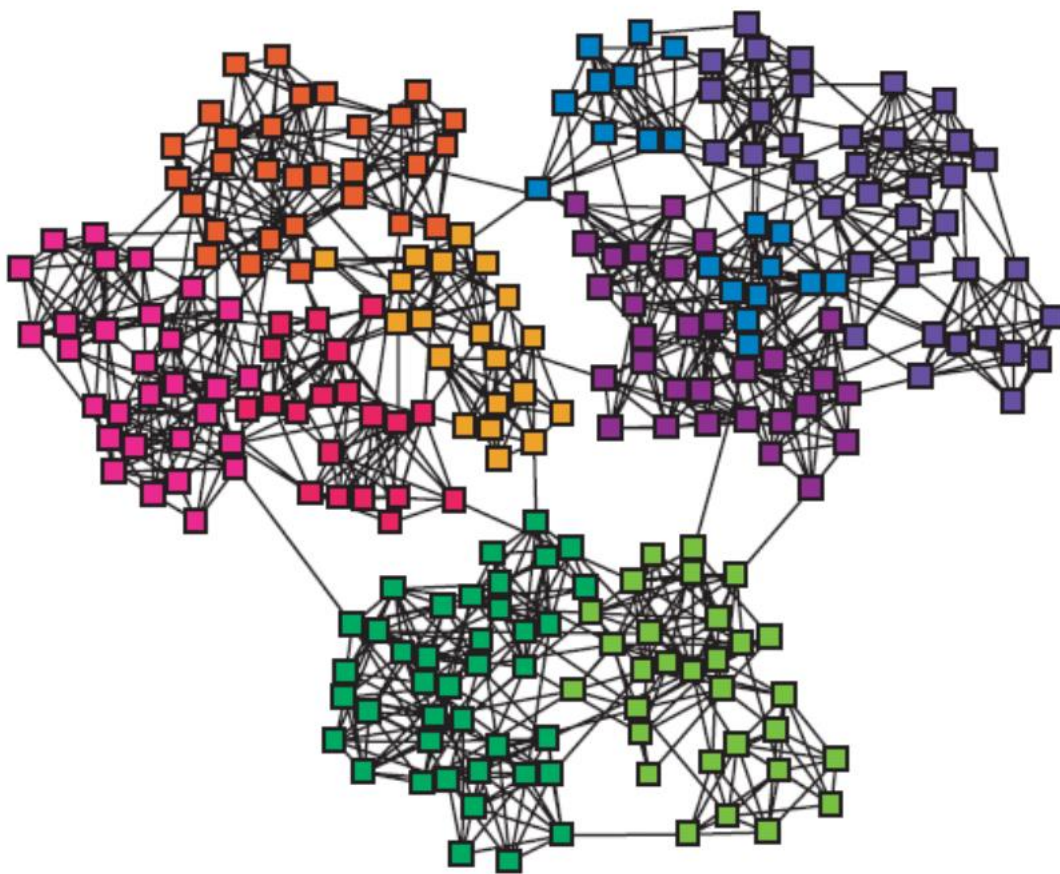


Figure 2: Graphes avec plusieurs communautés

Le clustering de graphes, aussi appelé détection de communautés, est une technique d'analyse des graphes visant à regrouper les nœuds en communautés, c'est-à-dire en sous-ensembles où les connexions internes sont denses, tandis que les liens avec le reste du graphe sont plus faibles. La détection de communautés a pour rôle de mettre en évidence ces groupes qui se sont formés implicitement, c'est-à-dire qui ne résultent pas d'un choix explicite [3]. Par exemple, dans une entreprise, les employés forment des communautés explicites (les services), mais certains vont collaborer plus souvent avec d'autres, que ce soit au sein du même service ou à cheval sur différents services, et former des communautés implicites [42].

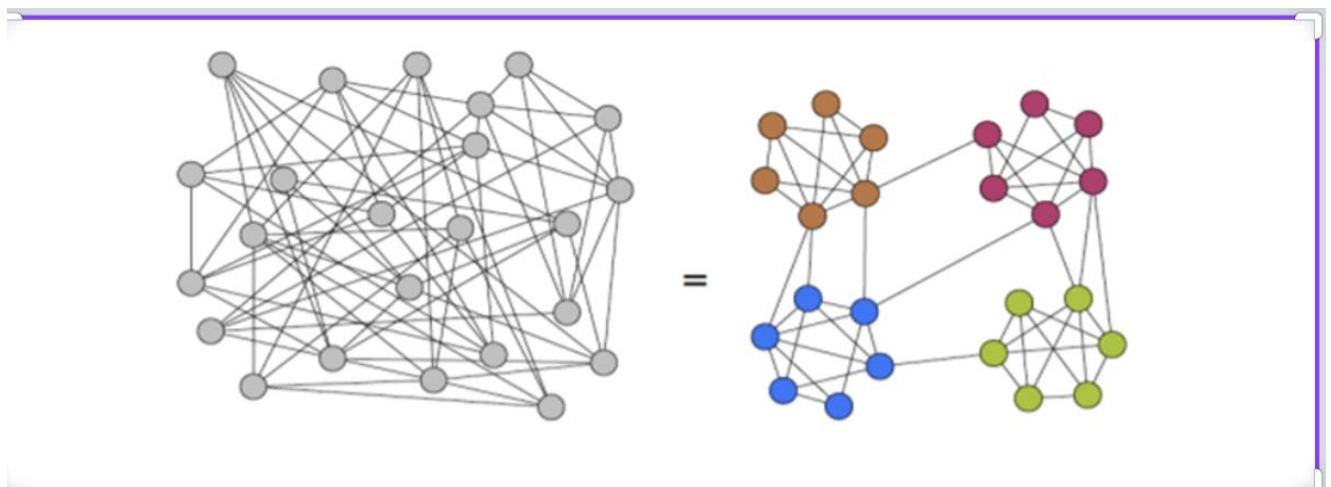


Figure 3: Détection de quatre communautés dans un réseau de graphe

L'intérêt de la détection de communautés est multiple : identifier des profils types, effectuer des actions ciblées, mieux ajuster les recommandations, réorganiser, identifier les acteurs centraux ou influents, etc. Différentes approches sont utilisées pour réaliser le clustering de graphes, telles que celles basées sur la connectivité, la similarité, la densité ou la modularité. Ces approches intègrent des algorithmes traditionnels tels que la méthode de Louvain ainsi que des techniques plus récentes d'apprentissage automatique et non supervisé.

La détection de communautés représente un champ de recherche en plein essor dans l'étude des réseaux complexes [41], offrant de nombreuses applications dans divers domaines tels que les réseaux sociaux, les réseaux de neurones, les réseaux de transport, les réseaux biologiques, et bien d'autres encore [40].

1.2 Problématique

L'étude du problème de la détection des communautés dans les réseaux devient, au fil des années, une question préoccupante. Ces dernières années, de nombreux algorithmes et méthodes ont été développés pour répondre à cette problématique, parmi lesquels les plus connus à ce jour sont : la méthode de classification, la méthode d'optimisation de la fonction objective, la méthode basée sur la théorie des graphes et la méthode basée sur les propriétés spectrales. L'une des méthodes les plus éminentes pour détecter les communautés est la classification spectrale [10]. Au cours des cinquante dernières années, de nombreux travaux de recherche ont exploré ces approches spectrales. Les travaux de [Dong, Z., Zhang, X., & Luo, B. \(2015\)](#) [14] ainsi que ceux de [Helene Berger \(2019\)](#) [5] sont des références dans ce domaine, reposant sur l'exploitation des méthodes de clustering spectral pour les réseaux dynamiques et l'utilisation des Graph Convolutional Networks (GCNs) pour la classification [49]. Ces approches exploitent efficacement les informations spectrales et les représentations graphiques pour des applications diverses, y compris l'analyse de réseaux biologiques et médicaux complexes [29].

Les défis liés à la détection de communautés dans les réseaux sont nombreux et variés :

- Définition floue des communautés : la notion même de ce qu'est une communauté dans un réseau peut être subjective et dépendante du contexte. Il n'existe pas de définition universelle acceptée, ce qui complique la comparaison et l'évaluation des algorithmes.
- Dépendance aux paramètres : de nombreux algorithmes nécessitent des paramètres tels que le nombre attendu de communautés ou des seuils de similarité, qui peuvent être difficiles à déterminer de manière objective [37].

- Sensibilité aux structures de réseau : certains algorithmes peuvent être efficaces sur des types spécifiques de réseaux (densément connectés ou clairsemés), mais peuvent échouer ou produire des résultats suboptimaux sur d'autres types de structures.
- Échelle : la complexité des algorithmes et le temps de calcul peuvent devenir prohibitifs à grande échelle, notamment sur des réseaux comportant des millions de nœuds et d'arêtes.
- Résolution de la modularité : l'optimisation de la modularité, souvent utilisée comme mesure pour détecter les communautés, peut conduire à des partitions sous-optimales lorsque le réseau possède des communautés de tailles très différentes ou des structures hiérarchiques.
- Détection des chevauchements : la détection de communautés chevauchantes (où un nœud peut appartenir à plusieurs communautés) reste un défi, bien que des avancées aient été réalisées avec des méthodes spécifiques.
- Interprétation des résultats : il peut être ardu d'interpréter et de valider les résultats des algorithmes de détection de communautés, surtout lorsque plusieurs partitions alternatives sont possibles.
- Évolution temporelle : les réseaux réels évoluent dans le temps, nécessitant une adaptation des méthodes de détection de communautés sans avoir à recalculer tout depuis le début [13].
- Bruit et données manquantes : la présence de bruit ou de données manquantes dans les réseaux réels peut compromettre la précision des résultats obtenus par les algorithmes.

Ces défis soulignent la complexité et la diversité des problèmes rencontrés dans le domaine de la détection de communautés, nécessitant des approches adaptées et souvent spécialisées pour répondre aux différentes nuances des réseaux réels.

Malgré la diversité des méthodes existantes, la détection de communautés dans les graphes demeure un défi. Chacune de ces méthodes présente ses avantages et ses inconvénients. Néanmoins, elles sont toutes limitées, et nous tenterons, à travers notre approche proposée dans ce mémoire, de résoudre ces limites.

1.3 Apport du mémoire

Parmi les contraintes rencontrées dans cette approche, dont la formulation mathématique reste complexe, notre étude propose une méthode hybride de détection de communautés dans les graphes, combinant l'approche de Louvain et la classification spectrale.

Cette méthode fusionne la classification spectrale avec l'algorithme de Louvain, offrant les caractéristiques suivantes :

- Classification spectrale : Utilise l'analyse des valeurs propres des matrices associées au réseau (comme la matrice de similarité ou la matrice de Laplace) pour réduire la dimensionnalité et identifier les structures communautaires en exploitant les propriétés spectrales [6].
- Algorithme de Louvain : Détecte les communautés en maximisant la modularité, une mesure de la qualité de la partition du réseau. Il fonctionne de manière itérative, améliorant la partition en regroupant les nœuds dans des communautés de manière locale et globale [47].

Combinaison des méthodes:

- Prétraitement spectral : La classification spectrale prétraite les données pour obtenir une représentation initiale des nœuds, facilitant la révélation de structures latentes et simplifiant le réseau [39].
- Partitionnement par Louvain : L'algorithme de Louvain est ensuite appliqué à cette représentation ou au réseau original pour affiner et optimiser la détection des communautés en maximisant la modularité [11].

Nos résultats seront comparés à ceux obtenus avec l'algorithme de Louvain et aux recherches menées par Mazouz de l'Université du Québec à Trois-Rivières. Cette méthode, fondée sur la classification spectrale, utilise l'agglomération conditionnelle des nœuds dans un graphe [31]. À partir des données de projection et des informations préliminaires, elle projette le graphe dans un espace bidimensionnel généré par deux vecteurs spécifiques.

1.4 Agencement du mémoire

Pour répondre efficacement à notre problème et atteindre les objectifs fixés, nous avons structuré notre mémoire comme suit :

1. Introduction : Le premier chapitre présente l'importance du problème, définit la question posée, expose nos objectifs et décrit le plan du mémoire. Nous abordons également les défis liés à la détection des communautés, en mettant en évidence les limites des méthodes existantes et en proposant nos propres idées pour contribuer à ce domaine.
2. État de la recherche : Le deuxième chapitre explore l'état actuel de la recherche sur la détection des communautés dans les réseaux. Nous analysons les défis spécifiques associés à cette tâche, examinons les solutions avancées dans la littérature scientifique et expliquons les concepts clés de la théorie des graphes nécessaires à la compréhension de notre étude.
3. Méthode proposée : Dans le troisième chapitre, nous détaillons la méthode que nous avons développée. Nous expliquons chaque étape de l'algorithme et les formules utilisées, afin de faciliter la compréhension de notre approche et du travail réalisé.
4. Mise en œuvre pratique : Le quatrième chapitre se concentre sur la mise en œuvre pratique des algorithmes développés. Ceux-ci sont programmés en langage R et disponibles sur la plateforme RStudio, ce qui permet de visualiser les résultats et de les comparer avec d'autres méthodes.
5. Conclusion : Enfin, la conclusion générale résume les découvertes clés, les contributions de notre étude et propose des perspectives pour la recherche future dans ce domaine.

2 ETAT DE L'ART

Ces dernières années, de nombreux algorithmes et méthodes ont été développés pour aborder cette thématique. Parmi les plus connus figurent la méthode de classification, l'optimisation de la fonction objective [35] et celle basée sur les propriétés spectrales, étroitement liées à la théorie des graphes [25]. Ce chapitre propose une revue des approches actuelles pour la détection de communautés dans les graphes. Avant d'entrer dans le vif du sujet, nous introduirons brièvement les notions de graphes ainsi que les concepts indispensables à la compréhension de cette problématique. Enfin, nous examinerons quelques méthodes en illustrant des approches récentes, avec leurs avantages et leurs limitations.

2.1 Postulat de base

Un graphe $G = (V, E)$ est une structure mathématique constituée de deux ensembles :

- V : Ensemble de sommets (ou nœuds).
- E : Ensemble d'arêtes, qui relient les sommets entre eux.

Pour un graphe non orienté :

- Chaque arête $e \in E$ est représentée par un ensemble $\{u, v\}$, où u sont des sommets et

$$u \neq v$$

Pour un graphe orienté :

- Chaque arête $e \in E$ est représentée par une paire ordonnée (u, v) , où u est le sommet de départ et v est le sommet d'arrivée.

2.1.1 Graphe non orienté

Un graphe fini $G = (V, E)$ est défini par un ensemble fini $V = \{v_1, v_2, \dots, v_n\}$ dont les éléments sont appelés sommets, et par un ensemble fini $E = \{e_1, e_2, \dots, e_m\}$ dont les éléments sont appelés arêtes [24]. Une arête e de l'ensemble E est définie par une paire non ordonnée de sommets, appelés extrémités de e . Si l'arête e relie les sommets a et b , on dit que ces sommets sont adjacents ou incidents avec e , et que l'arête e est incidente avec les sommets a et b . L'ordre d'un graphe est le nombre de sommets n dans ce graphe [7].

1. Quelques types de graphes

- Un graphe est dit simple (voir figure 4) s'il comporte au maximum une seule arête entre chaque paire de sommets et s'il ne contient pas de boucles (c'est-à-dire d'arêtes reliant un sommet à lui-même) [8]. En revanche, il est possible de former des multigraphes dans lesquels des boucles sont permises et où plusieurs arêtes peuvent relier les mêmes deux sommets [3].

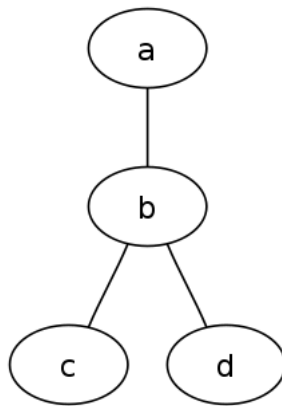


Figure 4: Graphe simple

- Un graphe est connexe s'il est possible, à partir de n'importe quel sommet, de rejoindre tous les autres en suivant les arêtes [13]. Un graphe non connexe se décompose en composantes connexes [16] (voir figure 5).

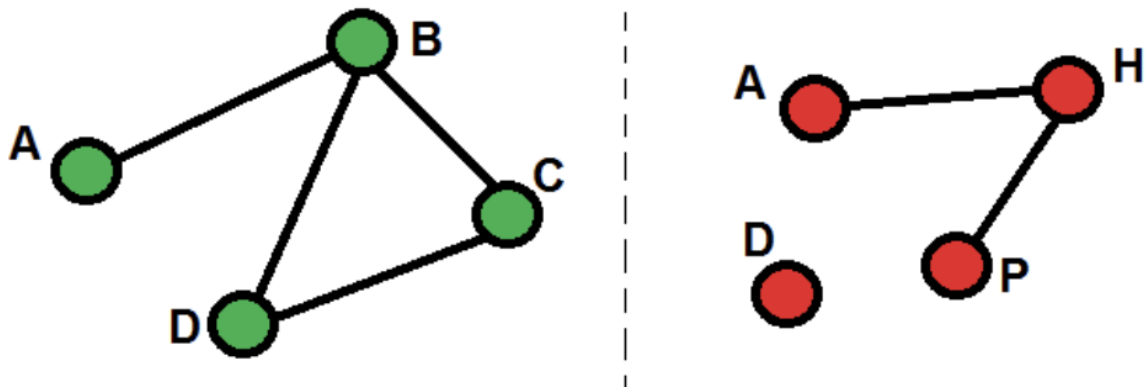


Figure 5: Graphe connexe et Graphe non-connexe

- Un graphe est complet si chaque sommet du graphe est relié directement à tous les autres sommets [2] (voir figure 6).

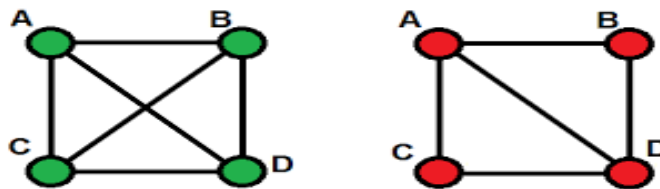


Figure 6: Graphe complet et Graphe non complet

- Un graphe est biparti si ses sommets peuvent être divisés en deux ensembles X et Y , de sorte que toutes les arêtes du graphe relient un sommet dans X à un sommet dans Y [25] (voir figure 7).

$X = \{1,2,3,4,5\}$ et $Y = \{A, B, C, E\}$

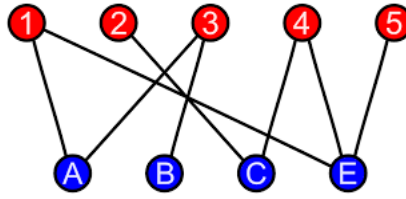
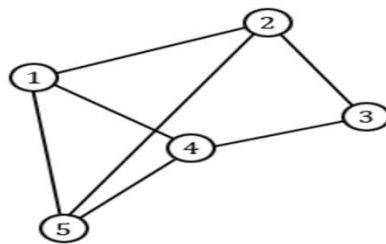


Figure 7: Graphe biparti

2. Représentations non graphiques d'un graphe non orienté

- **Matrice d'adjacence**

On peut représenter un graphe simple par une matrice d'adjacences. Une matrice $(n \times n)$ est un tableau de n lignes et n colonnes. (i, j) désigne l'intersection de la ligne i et de de la colonne j . Dans une matrice d'adjacences, les lignes et les colonnes représentent les sommets du graphe. Un « 1 » à la position (i, j) signifie que le sommet i est adjacent au sommet j [43].



$$\begin{pmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 5 \\ 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Figure 8: Graphe avec sa matrice d'adjacence

- **Liste d'adjacence d'un graphe non orienté**

On peut également représenter un graphe simple en fournissant, pour chaque sommet, la liste des sommets auxquels il est adjacent. Cette représentation est appelée liste d'adjacence.

La liste d'adjacence de la figure 8 est :

1: 2, 4, 5

2: 1, 3, 5

3: 2, 4

4: 1, 3, 5

5: 1, 2, 4

2.1.2 Graphe orienté (digraphe)

Dans certains cas, il peut être intéressant d'associer une orientation aux arêtes du graphe. Ainsi, au lieu d'avoir une arête non orientée $\{x, y\}$ reliant un sommet x à un sommet y , on peut avoir une arête orientée allant de x vers y ou de y vers x . [3] On parle alors d'arc et on utilise la notation (x, y) pour représenter un arc dirigé de x vers y . Un « chemin » dans un graphe orienté est une succession d'arcs qui permet de se rendre d'un sommet à un autre [24].

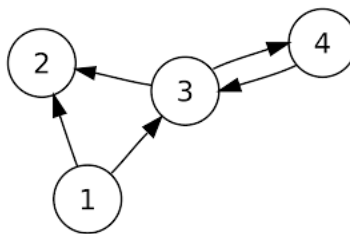


Figure 9: Graphe orienté

1. Degré d'un sommet d'un digraphe

Dans le cas d'un graphe orienté, on parle aussi du degré entrant d'un sommet $d^-(s)$, c'est-à-dire le nombre d'arcs dirigés vers le sommet s , et du degré sortant de ce sommet $d^+(s)$, c'est-à-dire le nombre d'arcs sortant de s . On a $deg(s) = d^+(s) + d^-(s)$,

$deg(s) = d^-(s) + d^+(s)$: le degré du sommet est la somme du degré sortant et du degré entrant [32].

2. Représentations non graphiques d'un graphe orienté

• Matrice d'adjacence

On peut représenter un digraphe par une matrice d'adjacences. Une matrice $(n \times n)$ est un tableau de n lignes et n colonnes. (i, j) désigne l'intersection de la ligne i et de la colonne j . Dans une matrice d'adjacences, les lignes et les colonnes représentent les sommets du graphe. Un « 1 » à la position (i, j) signifie qu'un arc part de i pour rejoindre j .

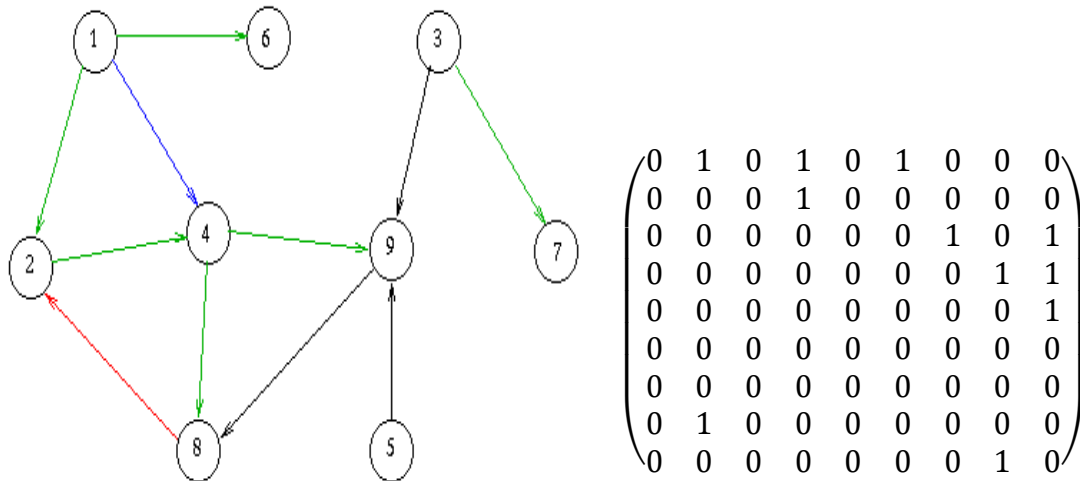


Figure 10: Graphe orienté avec sa matrice d'adjacence

- **Listes d'adjacences**

On peut également représenter un digraphe en fournissant, pour chaque sommet, la liste des sommets qu'on peut atteindre directement en suivant un arc (dans le sens de la flèche) [24].

1 : 6, 4, 2

2 : 4

3 : 7, 9

4 : 8, 9

5 : 9

6 :

7 :

8 : 2

9 : 8

2.2 Analyse spectrale de graphes

2.2.1 Définition

L'analyse spectrale des graphes est une branche des mathématiques et de l'informatique qui étudie les propriétés des graphes à travers les spectres de leurs matrices d'adjacence ou de Laplace [29]. À grands traits, l'analyse spectrale du laplacien d'un graphe se décompose en plusieurs étapes :

1. Transformation de la matrice d'adjacence : On calcule la matrice laplacienne en soustrayant la matrice des degrés de la matrice d'adjacence. La matrice laplacienne L est définie comme $L=D-A$, où D est la matrice des degrés et A est la matrice d'adjacence [8].

2. Décomposition en valeurs propres : On effectue la décomposition de la matrice laplacienne en ses valeurs propres. Ces valeurs propres fournissent des informations cruciales sur la structure et les propriétés globales du graphe, telles que la connectivité et la présence de communautés [38].

3. Application de méthodes de classification : On utilise les vecteurs propres associés aux valeurs propres pour appliquer des méthodes de classification. Ces vecteurs peuvent aider à identifier des structures de sous-graphes, des motifs récurrents ou des communautés au sein du graphe [48].

Cette approche permet d'étudier la connectivité, les motifs et la complexité des réseaux représentés par des graphes à travers leurs propriétés spectrales, offrant ainsi une compréhension approfondie de la structure et le comportement des systèmes complexes.

2.2.2 Exemple concret

Prenons l'exemple d'un graphe non orienté avec 7 sommets pour illustrer l'analyse spectrale à travers la matrice laplacienne.

Considérons le graphe G avec les sommets $\{A, B, C, D, E, F, G\}$ et les arêtes suivantes :

$\{A, B\}, \{A, C\}, \{B, C\}, \{B, D\}, \{C, D\}, \{C, E\}, \{D, E\}, \{D, F\}, \{E, F\}, \{E, G\}, \{F, G\}$

1. Transformation de la matrice d'adjacence

Pour comprendre l'analyse spectrale des graphes, il est important de connaître deux matrices principales associées à un graphe :

Matrice d'adjacence : C'est une matrice carrée $A = [a_{ij}]$ où a_{ij} est égal au nombre d'arêtes entre les sommets i et j . Pour un graphe non orienté, la matrice d'adjacence est symétrique. Pour un graphe orienté, elle peut ne pas l'être [43].

La matrice d'adjacence du graphe G est :

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

Matrice Laplacienne : La matrice Laplacienne d'un graphe est définie comme $L = D - A$, où A est la matrice d'adjacence, et D est la matrice diagonale des degrés des sommets (le degré d'un sommet étant le nombre d'arêtes connectées à ce sommet).

- Calcul des degrés :

$$D = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix} = \text{diag}(2, 3, 4, 4, 4, 3, 2)$$

- Matrice laplacienne L :

$$L = D - A = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 4 & -1 & -1 & 0 & 0 \\ 0 & -1 & -1 & 4 & -1 & -1 & 0 \\ 0 & 0 & -1 & -1 & 4 & -1 & -1 \\ 0 & 0 & 0 & -1 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix}$$

2. Décomposition en valeurs propres de L

Une fois la matrice Laplacienne L est obtenue, la décomposition en valeurs propres consiste à trouver les valeurs propres λ_i et les vecteurs propres correspondants v_i

- **Valeurs propres λ_i** : Ce sont les solutions de l'équation caractéristique $\det (L - \lambda_i I) = 0$,

Où I est la matrice identité $n \times n$

- **Vecteurs propres v_i** : Ce sont les vecteurs non nuls v_i tels que $Lv_i = \lambda_i v_i$

Pour le graphe G :

Les valeurs propres de L sont $\lambda_1 = 7.85$, $\lambda_2 = 5.94$, $\lambda_3 = 4$, $\lambda_4 = 2.73$, $\lambda_5 = 1.42$, $\lambda_6 = 0.97$, $\lambda_7 = 0.09$.

Les vecteurs propres correspondants peuvent être utilisés pour analyser la structure du graphe, identifier des sous-structures et des motifs.

3. Application d'une méthode de classification

Les méthodes de clustering permettent de regrouper des sommets en communautés ou en groupes similaires en utilisant des algorithmes adaptés aux représentations vectorielles des sommets [18]. Parmi les algorithmes de clustering courants, on trouve K-moyennes et DBSCAN. K-moyennes est souvent utilisé pour partitionner les sommets en un nombre fixe de clusters. Voici comment appliquer K-moyennes pour identifier les communautés dans un graphe en se basant sur les vecteurs propres obtenus par analyse spectrale [16].

Algorithme de K_moyennes

1. INITIALISER les centroïdes :

- Choisir k centroïdes aléatoires parmi les points de données.

2. RÉPÉTER jusqu'à convergence :

a. ATTRIBUER chaque point de données au cluster dont le centroïde est le plus proche :

- Calculer la distance entre chaque point et chaque centroïde.
- Assigner chaque point au cluster du centroïde le plus proche.

b. METTRE À JOUR les centroïdes :

- Pour chaque cluster, recalculer le centroïde en prenant la moyenne des points assignés à ce cluster.

3. TERMINER lorsque les centroïdes ne changent plus significativement ou le nombre maximum d'itérations est atteint.

4. RETOURNER les clusters et les centroïdes finaux.

2.3 Notions de communauté

Dans le cadre d'un graphe, une communauté désigne un ensemble de nœuds étroitement connectés entre eux mais moins reliés au reste du réseau [3]. Cette configuration peut inclure des individus interagissant fréquemment entre eux mais ayant moins d'échanges avec les autres [43].

Concepts de communauté dans les travaux de Fortunato, Hric:

- Cohésion interne et séparation Externe: Une communauté est définie comme un sous-graphe où le nombre d'arêtes internes (liens entre les membres de la communauté) est significativement plus élevé que le nombre d'arêtes externes (liens vers des nœuds en dehors de la communauté) [16].
- Communautés fortes et faibles : Une communauté est dite forte lorsque chaque nœud possède plus de liens internes (avec d'autres membres de la communauté) que de liens externes (vers des nœuds extérieurs). Elle est faible lorsque le total des liens internes de la communauté dépasse le total des liens externes la reliant au reste du réseau.
- Approche moderne : une communauté peut être définie comme un sous-graphe où les nœuds ont une probabilité plus élevée de se lier entre eux qu'avec des nœuds en dehors de la communauté [49]. Cette définition modernise le concept en introduisant des distinctions entre communautés fortes et faibles basées sur des probabilités d'arêtes [31].

En résumé, les communautés dans les graphes sont des sous-structures caractérisées par une forte cohésion interne et une faible connectivité externe. Les nuances dans les définitions reflètent divers critères de cohésion et de séparation, enrichissant ainsi l'analyse des structures communautaires dans les réseaux complexes.

Différents types de communautés sont utilisés pour représenter les structures et les interactions au sein des réseaux complexes [45]. Voici un résumé des principaux types de communautés :

- Communautés disjointes: Chaque nœud appartient à une seule communauté, les communautés étant mutuellement exclusives.
- Communautés chevauchantes : Les nœuds peuvent appartenir à plusieurs communautés simultanément, permettant de capturer les affiliations multiples dans le réseau.
- Communautés fortement connexes : Les nœuds au sein de ces communautés sont fortement interconnectés, formant des structures internes denses.
- Communautés faiblement Connexes : Les liens internes entre les nœuds sont moins denses, rendant la structure communautaire plus diffuse.

- Communautés basées sur les rôles : Regroupement de nœuds remplissant des rôles similaires ou complémentaires dans le réseau.
- Communautés multiplexes : Prise en compte de plusieurs types de relations entre les mêmes ensembles de nœuds, révélant des structures communautaires complexes.
- Communautés hiérarchiques: Organisées en niveaux ou en couches, avec des sous-communautés agrégées formant des communautés plus larges [6]..
- Communautés temporelles : Évolution dans le temps en réponse aux dynamiques du réseau, capturant les changements dans les affiliations communautaires.

Ces différents types de communautés permettent une analyse plus fine et une meilleure compréhension des structures et des dynamiques des réseaux complexes dans divers domaines d'application.

- **Communautés disjointes**

Une communauté disjointe est un sous-ensemble de nœuds dans un graphe où chaque nœud n'appartient qu'à une seule communauté. Cela signifie qu'il n'y a pas de chevauchement entre les communautés ; chaque nœud est exclusivement assigné à une seule entité communautaire. Chaque communauté disjointe est définie de manière que les nœuds internes partagent plus de liens entre eux qu'avec les nœuds extérieurs à la communauté. Cela crée des clusters ou des groupes compacts de nœuds fortement connectés à l'intérieur de chaque communauté [17].

Considérons un graphe G : dans cet exemple, nous avons trois communautés disjointes :

- **Communauté A:** jaune
- **Communauté B:** rouge
- **Communauté C:** mauve

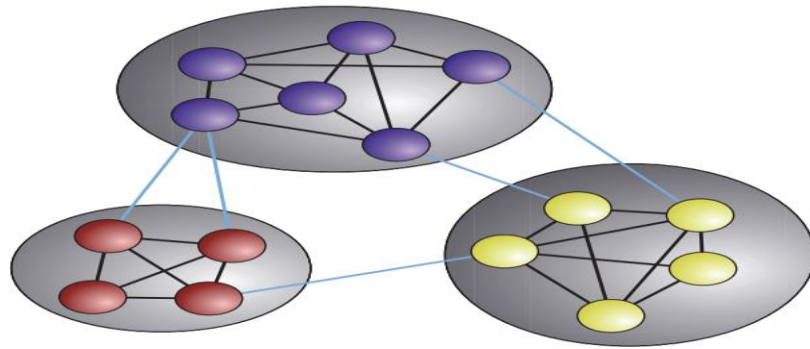


Figure 11: Exemple de graphe avec trois communautés disjointes

- **Communautés chevauchantes**

Une communauté chevauchante est un ensemble de nœuds dans un graphe où chaque nœud peut faire partie de plusieurs communautés différentes en même temps. Cela signifie que les frontières entre les communautés peuvent se chevaucher, permettant ainsi à un nœud d'avoir des affiliations diverses et potentiellement diversifiées au sein du réseau [14].

Les communautés chevauchantes capturent la complexité des relations dans les réseaux où les entités (nœuds) peuvent participer à plusieurs groupes ou sous-groupes selon différents critères ou types de connexions [42].

Imaginons un réseau social où les nœuds représentent des utilisateurs et les arêtes des relations (amitiés). Dans cet exemple :

- Le nœud A pourrait appartenir à la communauté X (amis proches), ainsi qu'à la communauté Y (collègues).
- Le nœud B pourrait appartenir à la communauté Y (collègues) et à la communauté Z (intérêts communs).

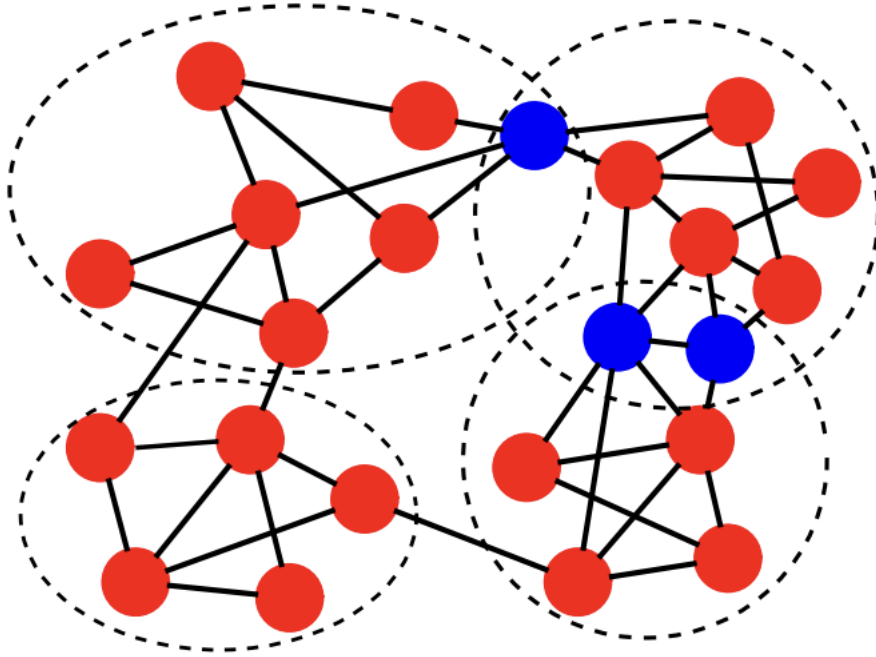


Figure 12: Exemple de communautés chevauchantes

- **Communautés fortes et faibles**

Selon la définition de [Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., & Parisi, D. \(2004\)](#) [40] une communauté faible est caractérisée par le fait que la somme des degrés internes de ses nœuds est supérieure à la somme des degrés externes. Une communauté est dite forte au sens de Hu et al si, pour tout sommet v appartenant à la communauté C , son degré interne est supérieur à son degré externe, soit :

$$d_{int}(v) > d_{ext}(v), \forall v \in C$$

$d_{int}(v)$ est le nombre d'arêtes reliant v aux autres sommets de C , $d_{ext}(v)$ est le nombre d'arêtes reliant v aux sommets extérieurs à C .

Cependant, trois des sous-graphes ne sont pas considérés comme des communautés fortes selon [Radicchi.](#), car certains sommets (indiqués en bleu) ont un degré externe plus élevé que leur degré interne (les arêtes internes et externes de ces sommets sont colorées respectivement en jaune et magenta).

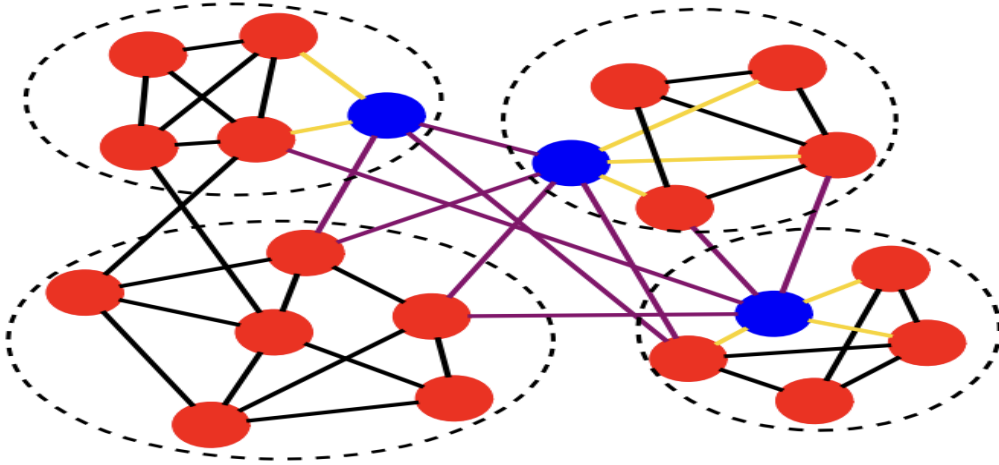


Figure 13: Exemple de Communautés fortes et faibles

2.4 Méthode de détection de communautés

La détection de communautés dans les graphes implique l'application d'algorithmes et de techniques visant à identifier des ensembles de nœuds fortement interconnectés entre eux au sein d'un réseau, par opposition aux connexions avec les nœuds extérieurs [3]. Nous allons voir les méthodes utilisées au fil des années, avec leurs avantages et limites.

2.4.1 Approche d'optimisation de la Modularité

Le problème de détection de communautés peut être ramené à un problème d'optimisation avec la définition d'une fonction objective de qualité d'une partition. Le critère de la modularité est la fonction objective la plus utilisée [30]. La modularité d'une partition mesure la différence entre la proportion de liens internes aux communautés et la même quantité dans un modèle nul où aucune structure communautaire n'est attendue [44].

La qualité pour une communauté c_i est donnée par :

$$\sum_{i,j \in c_i} \left(A_{ij} - \frac{d_i d_j}{2m} \right)$$

Où A_{ij} est l'élément de la matrice d'adjacence, d_i et d_j sont les degrés des nœuds i et j , m est le nombre total de liens dans le graphe.

La qualité d'une partition $Q(P)$ est souvent mesurée par des critères tels que la modularité, qui évalue la cohésion interne des groupes et leur séparation, et est donnée par la formule suivante :

$$Q(P) = \frac{1}{2m} \sum_{c \in P} \sum_{i,j \in c} \left(A_{ij} - \frac{d_i d_j}{2m} \right)$$

Le gain de modularité mesure l'amélioration de la modularité lorsqu'un sommet ou un groupe de sommets est déplacé d'une communauté à une autre dans une partition d'un graphe. Le gain de modularité $\Delta(Q)$ lors du déplacement d'un sommet i de la communauté c_1 à la communauté c_2 est donné par :

$$\Delta(Q) = \frac{1}{2m} \left[\sum_{j \in c_2} \left(A_{ij} - \frac{d_i d_j}{2m} \right) - \sum_{j \in c_1} \left(A_{ij} - \frac{d_i d_j}{2m} \right) \right]$$

Si $\Delta(Q) > 0$, le déplacement améliore la qualité de la partition (c'est-à-dire que la modularité augmente) et si $\Delta(Q) < 0$, le déplacement dégrade la partition (la modularité diminue).

Principe de l'algorithme de Louvain

- Initialisation des Communautés : Attribuez chaque nœud à sa propre communauté. Si le graphe a n nœuds, il y a initialement n communautés.
- Calcul du gain de modularité : Pour chaque nœud i , calculez le gain de modularité Q si i est déplacé de sa communauté C_i à une communauté voisine C_j .
- Déplacement du nœud : Déplacez le nœud i dans la communauté pour laquelle le gain de modularité est le plus élevé (à condition que ce gain soit positif).
- Répétition du processus : Répétez les étapes 2 et 3 pour tous les nœuds jusqu'à ce qu'aucun déplacement de nœud ne puisse améliorer la modularité.
- Création d'un nouveau réseau agrégé : Construisez un nouveau réseau où chaque nœud représente une communauté détectée lors de la phase 1.

- Répétition de l'optimisation : Appliquez la phase 1 et 2 sur le réseau agrégé nouvellement formé.
- Vérification de l'amélioration : Calculez la nouvelle modularité du réseau agrégé jusqu'à ce qu'il n'y ait plus d'amélioration de la modularité.
- Finalisation : Une fois la modularité maximale atteinte, l'algorithme retourne les communautés détectées.

Avantages	Limites
-Concept simple et intuitif	- NP-difficulté, coût computationnel élevé
-Applicable à divers types de graphes	-Sensibilité aux paramètres algorithmiques
-Identification de communautés fortement cohésives	-Risque de résultats suboptimaux
	- Tendance à identifier des communautés de tailles similaires
- Flexibilité dans le choix des algorithmes	

Tableau 1: Avantages et limites de l'approche d'optimisation

2.4.2 Approche du chemin aléatoire

L'approche du chemin aléatoire est une méthode heuristique utilisée pour détecter des communautés dans les graphes. Elle repose sur l'idée d'effectuer des marches aléatoires à partir de nœuds dans le graphe pour mesurer la similarité entre eux. Les nœuds qui sont souvent atteints ensemble par ces marches aléatoires sont regroupés en communautés, où les connexions internes sont plus fréquentes que les connexions externes. Cette méthode est flexible et efficace pour explorer des structures complexes de graphes, mais elle nécessite une sélection appropriée des paramètres et peut donner des résultats sensibles aux conditions initiales des marches aléatoires. Cette méthode a été développée par [Latapy, M., Pons, P., & Bouche, D. \(2008\)](#) [28].

Principe de l'algorithme du chemin aléatoire

Au début, $P_1 = \{\{v\}, v \in V\}$ et on calcule toutes les distances entre nœuds voisins. Puis à l'étape k :

- On choisit grâce à un critère de sélection deux communautés C_1 et C_2 parmi les communautés actuelles P_k .
- On les fusionne en une nouvelle communauté $C_3 = C_1 \cup C_2$ et on crée la nouvelle partition: $P_{k+1} = (P_k \setminus \{C_1; C_2\}) \cup \{C_3\}$
- On met à jour la distance entre les communautés.
- Ces étapes se répètent $n - 1$ fois (où n est le nombre initial de nœuds), jusqu'à ce que toutes les communautés soient fusionnées en une seule, correspondant à l'ensemble des nœuds du graphe.

Avantages	Limites
<ul style="list-style-type: none"> -Efficace pour les grands graphes -Adaptabilité à différentes structures de graphes -Identification de communautés de tailles variables - Intuitif dans l'interprétation des résultats 	<ul style="list-style-type: none"> - Sensible aux paramètres choisis - Complexité computationnelle pour de très grands graphes - Dépendance aux conditions initiales des marches aléatoires - Peut produire des résultats sensibles à la qualité des données

Tableau 2: Avantages et Limites de l'approche du chemin aléatoire

2.4.3 Méthode de classification spectrale

Le clustering spectral est une méthode non supervisée qui identifie des clusters de nœuds dans des graphes ou des individus dans des données individus/variables. Il repose sur la décomposition spectrale du Laplacien normalisé d'une matrice de similarité [11] . La laplacienne normalisée est une version de la matrice laplacienne qui prend en compte les degrés des sommets de manière à rendre les valeurs plus équilibrées. Elle est définie comme :

$$L_{norm} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$$

Où :

L est la matrice laplacienne du graphe, D est la matrice de degré du graphe, qui est une matrice diagonale avec les degrés des sommets en tant que valeurs diagonales, $D^{-\frac{1}{2}}$ est la matrice inverse de la racine carrée de D . Cette approche vise à partitionner les données (ou observations) en

groupes en utilisant un graphe où chaque sommet représente une donnée (ou observation) et chaque arête pondérée relie deux observations selon leur similarité.

Algorithme de la méthode de classification spectrale :

Entrées:

- Tableau de données $n \times p$
- K un noyau
- K le nombre de clusters.
- Calculer la matrice de similarités W sur les données en utilisant le noyau K
- Calculer le Laplacien normalisé L_{norm} à partir de W .
- Calculer les k premiers vecteurs propres $u_1, \dots, u_k$ de L_{norm} . On note U la matrice $n \times k$ qui les contient.
- Calculer la matrice T en normalisant les lignes de U : $t_{ij} = u_{ij} / \left(\sum_l u_{il}^2 \right)^{\frac{1}{2}}$
- Faire un k -moyennes avec les points $y_i, i=1, \dots, n$ (i-ème ligne de T) $\Rightarrow A_1, \dots, A_k$.

Sorties : clusters $C_1, \dots, C_k$ avec

$$C_j = \{i \mid y_i \in A_j\}$$

Avantages	Limites
<ul style="list-style-type: none"> -Capable de traiter des structures de données complexes -Robuste au bruit et aux valeurs aberrantes -Flexibilité dans le choix de la similarité -Interprétation intuitive des résultats - Adaptabilité à différents types de données 	<ul style="list-style-type: none"> -Sensible aux paramètres -Coût computationnel élevé pour le calcul des valeurs propres -Dépendance à la qualité de la construction du graphe de similarités -Résultats non déterministes (variation possible d'une exécution à l'autre) -Difficulté dans l'interprétation des clusters complexes et non clairement définis

Tableau 3: Avantages et Limites de l'approche de la classification spectrale

2.5 Conclusion

Ce chapitre nous a permis de mieux comprendre les graphes et les différentes méthodes pour détecter les communautés au sein de ces graphes. Nous avons examiné l'algorithme de Louvain, qui est efficace pour identifier des groupes en maximisant la modularité, et la classification spectrale, qui utilise les propriétés des vecteurs propres de la matrice Laplacienne pour révéler des structures cachées. Nous avons discuté des avantages et des limites de ces techniques. Cette connaissance nous prépare à utiliser ces approches dans le chapitre suivant pour améliorer notre méthode d'analyse des communautés dans les réseaux complexes.

3 MÉTHODE PROPOSÉE: APPROCHE HYBRIDE

Le présent chapitre introduit une nouvelle approche de recherche pour la détection de communautés dans les réseaux. De nombreuses méthodes existent déjà dans ce domaine [29], basées sur les travaux de recherche antérieurs qui ont démontré leur qualité et leur pertinence.

Dans notre étude, nous explorons deux approches principales : l'algorithme de Louvain et la classification spectrale. Dans ce chapitre, nous détaillerons chaque étape de notre approche, mettant en lumière les mécanismes utilisés et les résultats attendus.

3.1 Présentation

Notre méthode hybride combine l'algorithme de Louvain, reconnu pour sa rapidité, avec la classification spectrale, qui offre une précision dans la séparation des clusters. Cette approche vise à surmonter les limites individuelles de chaque méthode pour fournir une détection plus précise et efficace des communautés.

3.2 Principe de la méthode hybride

Notre approche hybride combine deux méthodes complémentaires pour améliorer la détection de communautés dans les réseaux : l'algorithme de Louvain et la classification spectrale.

- **Algorithme de Louvain**

Cette méthode repose sur l'optimisation de la modularité, qui mesure la densité des connexions à l'intérieur des communautés par rapport à celles entre elles. Le processus se déroule en deux phases :

Phase 1 : Attribution des nœuds à des communautés basées sur les connexions locales, permettant une détection rapide des communautés initiales [41].

Phase 2: Regroupement des communautés détectées pour maximiser la modularité globale [12].

L'algorithme de Louvain est particulièrement adapté pour traiter des réseaux de grande taille en raison de sa rapidité, ce qui le rend idéal pour des applications nécessitant une réponse rapide, comme l'analyse en temps réel des réseaux sociaux [35].

- **Classification spectrale**

Cette méthode utilise les valeurs propres de la matrice de Laplace du graphe pour déterminer des clusters, fournissant des informations sur la structure globale du réseau. Elle permet de :

Affiner les frontières : en se basant sur les résultats de Louvain, la classification spectrale améliore la définition des clusters et réduit le chevauchement entre communautés.

Améliorer la précision : en exploitant les propriétés mathématiques du graphe, cette étape offre une séparation plus nette entre les communautés.

En combinant ces deux approches, notre méthode hybride propose une solution robuste et efficace pour la détection de communautés, optimisant la rapidité de Louvain tout en rehaussant la précision des résultats par l'analyse spectrale. Cela permet de mieux appréhender la complexité des réseaux interconnectés et de révéler des structures communautaires significatives qui pourraient passer inaperçues avec une méthode unique.

3.3 Fonctionnement

La mise en œuvre de notre méthode se déroule en plusieurs étapes :

1. Détection initiale avec l'algorithme de Louvain : Le réseau est d'abord analysé par l'algorithme de Louvain, qui génère des communautés initiales. Ces communautés sont souvent des regroupements denses de nœuds qui peuvent ne pas être parfaitement séparés [41].
2. Analyse spectrale des communautés : Les communautés obtenues sont ensuite soumises à une analyse spectrale. Nous calculons les valeurs propres de la matrice de Laplace, qui nous permettent d'obtenir un vecteur représentant chaque communauté dans un espace de caractéristiques.
3. Affinement des frontières de communauté : À l'aide des valeurs propres et des vecteurs propres, nous identifions des partitions plus précises, en affinant les frontières entre les communautés.

4. Clustering spectral à l'aide de l'algorithme k-moyennes : identifie des groupes dans des données complexes en utilisant les valeurs propres d'un graphe. Après réduction de dimensionnalité, l'algorithme k-moyennes partitionne les clusters sur les données projetées.

Entrées :

Les entrées de notre méthode se composent des éléments suivants :

- Matrice d'adjacence :
 - Type : Matrice binaire ou pondérée.
 - Description : Représente les connexions entre les sommets du graphe. Dans notre étude, une matrice d'adjacence aléatoire est générée avec 15 sommets.
- Paramètres de fonction:
 - Options de Visualisation : Paramètres pour personnaliser l'affichage du graphe, comme la taille des sommets et la couleur des étiquettes.
- Bibliothèques R:
 - Les bibliothèques igraph, RSpectra, et Matrix doivent être chargées pour utiliser les fonctions d'analyse et de visualisation.

Sorties :

Les sorties de notre méthode comprennent :

- Graphes visualisés:
 - Description : Affichages graphiques du graphe d'origine et des graphes colorés par communauté. Cela permet d'identifier visuellement la structure et les groupes dans le graphe.
- Matrices:
 - Matrice d'Adjacence : La matrice utilisée pour créer le graphe.
 - Matrice Laplacienne : Calculée à partir de la matrice d'adjacence, elle est utilisée pour la classification spectrale.
- Communautés détectées:
 - Description : Le nombre de communautés identifiées par l'algorithme de Louvain et les membres de chaque communauté.
- Résultats de clustering:

- Description : Les résultats du clustering spectral, indiquant à quel cluster chaque sommet appartient.
- Valeurs et vecteurs propres:
 - Description : Les valeurs propres et les vecteurs propres associés à la matrice laplacienne, utilisés dans la classification spectrale

Prenons le graphe G avec sa matrice d'Adjacence

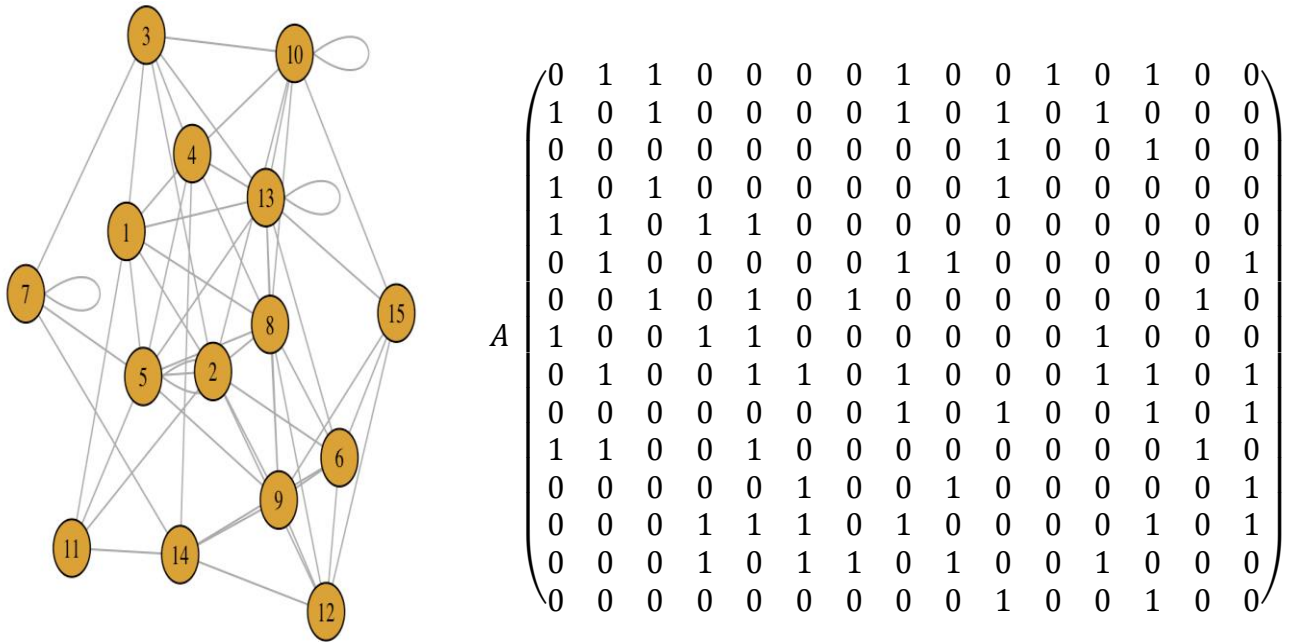


Figure 14: Graphe G avec sa matrice d'adjacence (A)

3.3.1 Détection initiale des communautés avec Louvain

L'algorithme de Louvain consiste à:

- Optimiser localement la modularité en déplaçant les sommets.
- Construire un nouveau niveau avec les communautés comme nœuds.
- Répéter le processus jusqu'à ce que la modularité ne puisse plus être améliorée.[6]

1. Calcul des paramètres

- **Calcul des degrés des sommets**

Le degré d'un sommet v_i est la somme des éléments de la i -ème ligne de la matrice d'adjacence.

$$k_i = \sum_j A_{ij}$$

Calculons le degré pour les 15 sommets

- Degré (v_1)= $0+1+1+0+0+0+0+1+0+0+1+0+1+0+0=5$
- Degré (v_2)= $1+0+1+0+0+0+0+1+0+1+0+1+0+0+0=5$
- Degré (v_3)= $0+0+0+0+0+0+0+0+0+1+0+0+1+0+0=2$
- Degré (v_4)= $1+0+1+0+0+0+0+0+1+0+0+0+0+0+0=3$
- Degré (v_5)= $1+1+0+1+1+0+0+0+0+0+0+0+0+0+0=4$
- Degré (v_6)= $0+1+0+0+0+0+0+0+1+1+0+0+0+0+1=4$
- Degré (v_7)= $0+0+1+0+1+0+1+0+0+0+0+0+0+1+0=4$
- Degré (v_8)= $1+0+0+1+0+1+0+0+0+0+0+1+0+0+0=4$
- Degré (v_9)= $0+1+0+0+1+1+0+1+0+0+0+1+1+0+1=7$
- Degré (v_{10})= $0+0+0+0+0+0+0+1+0+1+0+0+1+0+1=4$
- Degré (v_{11})= $1+1+0+0+1+0+0+0+0+0+0+0+0+1+0=4$
- Degré (v_{12})= $0+0+0+0+0+1+0+0+1+0+0+0+0+0+1=3$
- Degré (v_{13})= $0+0+1+1+1+1+0+1+0+0+0+0+1+0+1=6$
- Degré (v_{14})= $0+0+0+1+0+1+1+0+1+0+0+1+0+0+0=5$
- Degré (v_{15})= $0+0+0+0+0+0+0+0+1+0+0+1+0+0+0=2$

- **Calcul du nombre total d'arêtes**

Le nombre total d'arêtes m est :

$$m = \frac{1}{2} \sum_{i,j} A_{ij}$$

$$\text{Nombre total d'arêtes : } \frac{1}{2} \times 58 = 29$$

2. Calcul de la modularité initiale

- **Initialisation**

Chaque nœud commence dans sa propre communauté :

Communautés initiales : $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \{11\}, \{12\}, \{13\}, \{14\}, \{15\}$

- **Calcul de la modularité pour chaque nœud**

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

- A_{ij} est le poids de l'arête entre les nœuds i et j (1 pour les arêtes présentes, 0 pour les autres).
- k_i et k_j sont les degrés des nœuds i et j .
- m est le nombre total d'arêtes.
- $\delta(c_i, c_j)$ est 1 si i et j sont dans la même communauté, sinon 0

Dans cette partition initiale où chaque sommet est dans sa propre communauté, $\delta(c_i, c_j) = 0$ pour tous les $i \neq j$. Donc la modularité initiale est :

$$Q = 0$$

3. Optimisation locale

L'optimisation locale est une méthode utilisée pour améliorer la structure des communautés dans un réseau afin de maximiser la modularité. La modularité mesure la densité des connexions à l'intérieur des communautés comparée aux connexions entre communautés [16]. Un score de modularité élevé indique une meilleure séparation entre les communautés.

4. Déplacement des sommets

Pour améliorer la modularité, l'une des techniques consiste à déplacer les sommets d'une communauté à une autre [41]. Cela implique de vérifier, pour chaque sommet, s'il serait mieux positionné dans une autre communauté. Le déplacement de sommets se fait en évaluant si ce changement augmenterait la modularité totale du réseau.

Concrètement, cela revient à :

- Prendre un sommet donné.
- L'extraire temporairement de sa communauté actuelle.
- Calculer la variation de modularité si ce sommet est placé dans une autre communauté.
- Si cette nouvelle disposition augmente la modularité, on valide le déplacement du sommet. Sinon, il reste dans sa communauté d'origine.

5. Créer une nouvelle partition hypothétique

Supposons que nous avons une partition où les sommets sont regroupés en deux communautés :

- **Communauté 1:** {1,2,3,4,5,6,7}
- **Communauté 2:** {8,9,10,11,12,13,14,15}

6. Calculer la modularité pour cette partition

La modularité est calculée et faudra d'abord calculer le nombre d'arête comme vu dans la première étape.

Et pour finir répéter le processus jusqu'à ce que la modularité ne puisse plus être améliorée.

3.3.2 Construction de la matrice Laplacienne

La matrice Laplacienne L est définie comme la différence entre la matrice des degrés et la matrice d'adjacence : $L=D-A$

$$\text{Matrice des degrés (D)} = \begin{pmatrix} 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \end{pmatrix}$$

$$L = \begin{pmatrix} 7 & -1 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 \\ -1 & 9 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ -1 & 0 & -1 & 7 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & -1 & 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 7 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & -1 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & -1 & -1 & 0 & 0 & 9 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & -1 & 0 & -1 & 8 & 0 & 0 & -1 & -1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 7 & 0 & 0 & -1 & 0 & -1 \\ -1 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 6 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & -1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 10 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & -1 & 0 & -1 & 0 & 0 & -1 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 5 \end{pmatrix}$$

La matrice D est diagonale, où chaque entrée diagonale représente le nombre de connexions (ou degré) de chaque nœud dans le graphe. Par exemple, si un nœud a 7 connexions, la valeur correspondante sur la diagonale sera 7, tandis que les autres entrées de la ligne/colonne de ce nœud seront 0 [18].

Après avoir soustrait la matrice d'adjacence (qui indique les connexions entre les nœuds), on obtient la matrice Laplacienne L. Elle encode à la fois les informations sur le degré des nœuds et la structure du graphe. Les éléments diagonaux correspondent toujours au degré du nœud, et les éléments non diagonaux sont généralement des valeurs négatives (-1) lorsqu'il existe une connexion entre deux nœuds dans le graphe.

3.3.3 Classification spectrale sur la matrice Laplacienne

Nous allons effectuer un clustering spectral en utilisant les vecteurs propres de la matrice Laplacienne.

1. Décomposition spectrale

Extraction des valeurs et vecteurs propres : Nous calculons les valeurs propres et les vecteurs propres de la matrice Laplacienne L . Les valeurs propres, solutions de l'équation : $\det (L-\lambda_i)=0$ (où i est la matrice identité), indiquent des caractéristiques importantes de la matrice. Les vecteurs propres v_i associés, satisfaisant $Lv_i=\lambda_i v_i$, montrent comment les données sont structurées, car ils conservent leur direction sous l'action de L .

Valeurs propres λ_i : Les valeurs propres sont les solutions de l'équation caractéristique : $\det (L-\lambda_i)=0$ où i représente une matrice carrée où tous les éléments de la diagonale principale sont 1 et tous les autres éléments sont 0. La condition $\det (L-\lambda_i)=0$ signifie que la matrice $L-\lambda_i$ n'est pas inversible. Cela se produit lorsque λ est une valeur propre de L .

Vecteurs propres v_i : Les vecteurs propres sont les vecteurs non nuls v_i satisfaisant l'équation suivante : $Lv_i=\lambda_i v_i$ dans cette équation, Lv_i produit un nouveau vecteur qui est égal à $\lambda_i v_i$. Ici, λ_i est la valeur propre associée au vecteur propre v_i . Cela montre que l'effet de L sur v_i est simplement de l'étirer ou de le rétrécir par le facteur λ_i . Cela signifie que l'application de la matrice Laplacienne L sur le vecteur propre v_i donne un résultat qui est proportionnel à v_i lui-même, avec λ_i comme coefficient de proportionnalité

2. Clustering k-moyennes

Nous appliquons l'algorithme k-moyennes sur les vecteurs propres extraits afin de diviser les sommets en un certain nombre de groupes, appelés num_clusters.

- **Décomposition spectrale**

La décomposition spectrale décompose la matrice Laplacienne d'un graphe en ses valeurs et vecteurs propres, permettant une analyse approfondie de la structure du graphe. Les vecteurs propres associés aux petites valeurs propres capturent les principales structures sous-jacentes du graphe et sont utilisés pour effectuer le clustering spectral, ce qui aide à révéler les communautés ou les groupes significatifs dans le graphe.

La décomposition spectrale de la matrice Laplacienne L consiste à résoudre l'équation : $Lv_i = \lambda_i v_i$

-Après avoir effectué les calculs dans R, les valeurs propres obtenues sont les suivantes :

$$\text{Valeurs propres} = \begin{pmatrix} 11.7456907 \\ 10.6298808 \\ 10.4940834 \\ 9.9816949 \\ 8.7149687 \\ 8.3497508 \\ 7.4258454 \\ 7.3641730 \\ 6.6913994 \\ 6.3186409 \\ 5.4308438 \\ 3.9410654 \\ 3.6228413 \\ 3.0553894 \\ 0.2337321 \end{pmatrix}$$

Dans nos résultats, les valeurs propres indiquent l'importance des nouvelles directions principales dans la distribution de la variance des données. La première valeur propre, 11.745 , suggère que la première direction principale capture la plus grande part de la variance, c'est-à-dire qu'elle représente une composante clé des données initiales. Chaque valeur propre qui suit diminue progressivement, signifiant que les directions associées deviennent moins importantes en termes de variance capturée.

-Après avoir effectué les calculs dans R, les vecteurs propres obtenus sont les suivantes :

```
[1] "Valeurs propres :"
```

[1]	11.7456907	10.6298808	10.4940834	9.9816949	8.7149687	8.3497508	7.4258454	7.3641730
-----	------------	------------	------------	-----------	-----------	-----------	-----------	-----------

```
[9] 6.6913994 6.3186409 5.4308438 3.9410654 3.6228413 3.0553894 0.2337321
```

```
[1] "Vecteurs propres :"
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]
[1,]	-0.138236219	-0.171359633	0.144228579	-0.07083216	0.544513490	-0.2822804746	0.33159212
[2,]	0.463984795	0.073019638	0.099597324	0.70028099	0.163817048	0.0675657859	-0.17315902
[3,]	-0.159516411	0.147418662	-0.091794083	-0.06445373	-0.163992394	0.2173498765	-0.07578923
[4,]	0.023889685	-0.100605644	0.318044096	0.22797473	-0.503063758	-0.1013987525	0.27535140
[5,]	-0.325990266	-0.111494328	-0.695366945	0.22899349	-0.146927009	0.2634247052	0.03795273
[6,]	-0.207185682	-0.169838601	-0.004467137	0.03444136	-0.225908369	-0.5167259765	-0.57058545
[7,]	0.055687848	-0.027373359	0.153536591	-0.03689123	-0.014676866	-0.1297074046	0.01552633
[8,]	-0.055001832	0.811275847	0.044454572	-0.26569373	-0.005620168	-0.0801541644	-0.05441135
[9,]	-0.163916210	-0.205635252	0.457956713	-0.16373848	0.051188742	0.6830458681	-0.18400449
[10,]	-0.204416345	-0.248917715	-0.022138968	-0.10089096	0.332993327	-0.1094945730	-0.24293846
[11,]	-0.006961825	0.009694978	0.101855705	-0.15289556	-0.199688261	-0.0303765268	-0.05284288
[12,]	-0.012733929	-0.180157263	-0.083535145	-0.13239250	-0.139866365	-0.1008541364	0.53844794
[13,]	0.715622187	-0.207294114	-0.280737355	-0.46300629	-0.078349029	0.0445973726	-0.09830865
[14,]	0.054165832	0.145557774	-0.209918405	0.05613230	0.380120364	0.0834203051	-0.01535430
[15,]	-0.018881687	0.179727242	-0.012209153	0.16572409	0.016135182	-0.0001697305	0.22977107

	[,8]	[,9]	[,10]	[,11]	[,12]	[,13]	[,14]
[1,]	-0.36631502	-0.19617077	-0.22663831	-0.07415194	-0.29596445	-0.11691587	-0.196549600
[2,]	0.12385582	0.16748889	-0.28719326	-0.06674195	-0.07897698	-0.09590792	-0.009112453
[3,]	-0.38005457	0.62688664	0.03190691	0.05008423	-0.36054837	0.28330722	-0.211945133
[4,]	0.07648840	-0.26671148	0.43512789	-0.24235450	-0.29536048	0.08266264	-0.102883301
[5,]	0.08457402	-0.35438782	-0.20123324	-0.03024998	-0.04712966	-0.02808147	-0.146453961
[6,]	-0.27660691	-0.02748920	-0.09522638	-0.14164936	0.16545209	-0.01706053	0.270664188
[7,]	0.13159373	-0.11984239	-0.19308341	0.21281292	0.43176542	0.63509668	-0.457646370
[8,]	0.22832682	-0.19048696	-0.18619466	-0.20057931	-0.12916082	-0.01198222	0.080916786
[9,]	-0.05581282	-0.16908954	-0.13476113	-0.11043767	0.12343104	-0.01911011	0.200496741
[10,]	0.64331922	0.14887102	0.25074253	0.17174098	-0.32210497	0.11302552	0.076934545
[11,]	0.09069352	0.12373666	0.04228658	0.34628530	0.18754447	-0.68034689	-0.450994867
[12,]	0.23448242	0.42479491	-0.25095055	-0.24925470	0.27894132	-0.03854774	0.322761317
[13,]	-0.09048197	-0.16706434	0.01087770	0.02720024	-0.18170405	0.05550518	0.065161304
[14,]	-0.14722351	0.05004492	0.61701742	-0.32433629	0.43312394	-0.01569351	-0.073898497
[15,]	-0.19241398	-0.12417106	0.16632111	0.70187971	0.06045266	0.06812019	0.481339615
[,15]							
[1,]	-0.2648545						
[2,]	-0.2680335						
[3,]	-0.2534358						
[4,]	-0.2594160						
[5,]	-0.2330101						
[6,]	-0.2758451						
[7,]	-0.2011855						
[8,]	-0.2648160						
[9,]	-0.2709587						
[10,]	-0.2299083						
[11,]	-0.2753842						
[12,]	-0.2814411						
[13,]	-0.2379807						
[14,]	-0.2712726						
[15,]	-0.2719390						

Figure 15: Résultats des vecteurs propres dans R

Les vecteurs propres, associés à chaque valeur propre, décrivent la contribution des variables d'origine à ces nouvelles directions principales. Par exemple, dans la première colonne des vecteurs propres, qui correspond à la première direction (ou première composante principale), la deuxième variable a un coefficient élevé de **0.4639**, ce qui signifie qu'elle contribue fortement à cette direction. Cela indique que la deuxième variable joue un rôle majeur dans la variance capturée par la première composante principale. En revanche, des variables comme la quatrième, avec un coefficient de seulement **0.0238**, contribuent beaucoup moins à cette direction.

Ainsi, ces résultats montrent que les premières directions principales, associées aux plus grandes valeurs propres, capturent la majorité de l'information (variance) dans nos données. Ils nous permettent également d'identifier quelles variables dominent dans chaque composante principale, nous aidant à mieux comprendre quelles dimensions ou variables d'origine sont les plus influentes

dans l'explication de la structure globale des données. En pratique, cela nous permettrait de réduire la dimension des données en conservant les composantes principales les plus importantes, tout en perdant peu d'information.

- **Réduire les dimensions avec les vecteurs propres associés aux plus petites valeurs propres**

Une fois les vecteurs propres obtenus, on peut utiliser les premières colonnes de la matrice des vecteurs propres (correspondant aux plus petites valeurs propres) pour le clustering. Ces vecteurs représentent des coordonnées dans un espace réduit qui préservent la structure importante du graphe.

-0.138236219	-0.171359633	0.144228579
0.463984795	0.073019638	0.099597324
-0.159516411	0.147418662	-0.091794083
0.023889685	-0.100605644	0.318044096
-0.325990266	-0.111494328	-0.695366945
-0.207185682	-0.169838601	-0.004467137
0.055687848	-0.027373359	0.153536591
-0.055001832	0.811275847	0.044454572
0.163916210	-0.205635252	0.457956713
-0.204416345	-0.248917715	-0.022138968
-0.006961825	-0.009694978	0.101855705
-0.012733929	- 0.180157263	-0.083535145
0.715622187	-0.207294114	-0.280737355
0.054165832	0.145557774	-0.209918405
-0.018881687	0.179727242	-0.012209153

Tableau 4: Vecteurs propres des 3 plus petites valeurs propres

3.3.4 Clustering spectral à l'aide de l'algorithme k-moyennes

Ces vecteurs sont ensuite utilisés comme caractéristiques d'entrée pour un algorithme de clustering tel que k-moyennes. Le clustering k-moyennes partitionne ces vecteurs en `num_clusters` groupes, ce qui permet de découvrir les communautés ou les clusters dans le graphe.

Rappel de l'algorithme k-moyennes

Algorithme k-moyennes

Entrée:

- Ensemble de données X
- Nombre de clusters k

Initialisation:

- Sélectionner k centres initiaux aléatoires

Tant qu'il n'y a pas convergence:

- Assigner chaque point au centre de cluster le plus proche
- Recalculer les centres des clusters comme la moyenne des points assignés
- Vérifier la convergence (les centres changent-ils significativement ?)

Sortie:

- Centres des clusters finaux
- Assignations des points aux clusters

Pour appliquer l'algorithme k-moyennes, nous devons :

-Choisir le nombre de clusters k (ici, $k=3$).

L'algorithme k-moyennes commence par sélectionner aléatoirement k points comme centres initiaux (centroïdes). Ensuite, chaque point de données est assigné au cluster dont le centroïde est le plus proche. Une fois l'assignation faite, les centres des clusters sont recalculés en prenant la moyenne des points assignés à chaque cluster. Ce processus d'assignation et de recalcul des centroïdes se répète jusqu'à ce que les centres ne changent plus ou changent très peu, signalant la convergence. L'algorithme renvoie alors les positions finales des centres et les assignations des points aux clusters. .

1. Initialisation

Pour la simplicité, nous pouvons initialiser les centres des clusters aléatoirement ou en utilisant une méthode comme k-moyennes : Choix des centroïdes dans R studio

Centre 1 = [-0.32599027 -0.1114943 -0.6953669]

Centre 2 = [-0.09037657 -0.1051971 0.1081874]

Centre 3 = [0.23197786 0.2004573 -0.0717626]

2. Calcul des distances euclidiennes

Il faudra Calculer les distances euclidiennes entre chaque point et les centres initiaux. Pour Assignez le point au cluster dont le centre est le plus proche

La distance euclidienne entre deux points $(x1, y1)$ et $(x2, y2)$ est défini par la formule:

$$D = \sqrt{(X1 - X2)^2 + (Y1 - Y2)^2 + (z1^2 - z2)^2}$$

En R, les calculs montrent que les points sont attribués aux clusters comme suit :

- Cluster 1: 5
- Cluster 2: 1, 3, 4, 6, 7, 9, 10, 11, 12
- Cluster 3: 2, 8, 13, 14, 15

L'algorithme continue à itérer entre les étapes d'assignation des points et de mise à jour des centres jusqu'à ce que les centres des clusters ne changent plus ou que le changement soit en dessous d'un seuil prédéfini. L'algorithme continue à itérer entre les étapes d'assignation des points et de mise à jour des centres jusqu'à ce que les centres des clusters ne changent plus ou que le changement soit en dessous d'un seuil prédéfini.

3.4 Visualisation des communautés détectées par l'algorithme Hybride

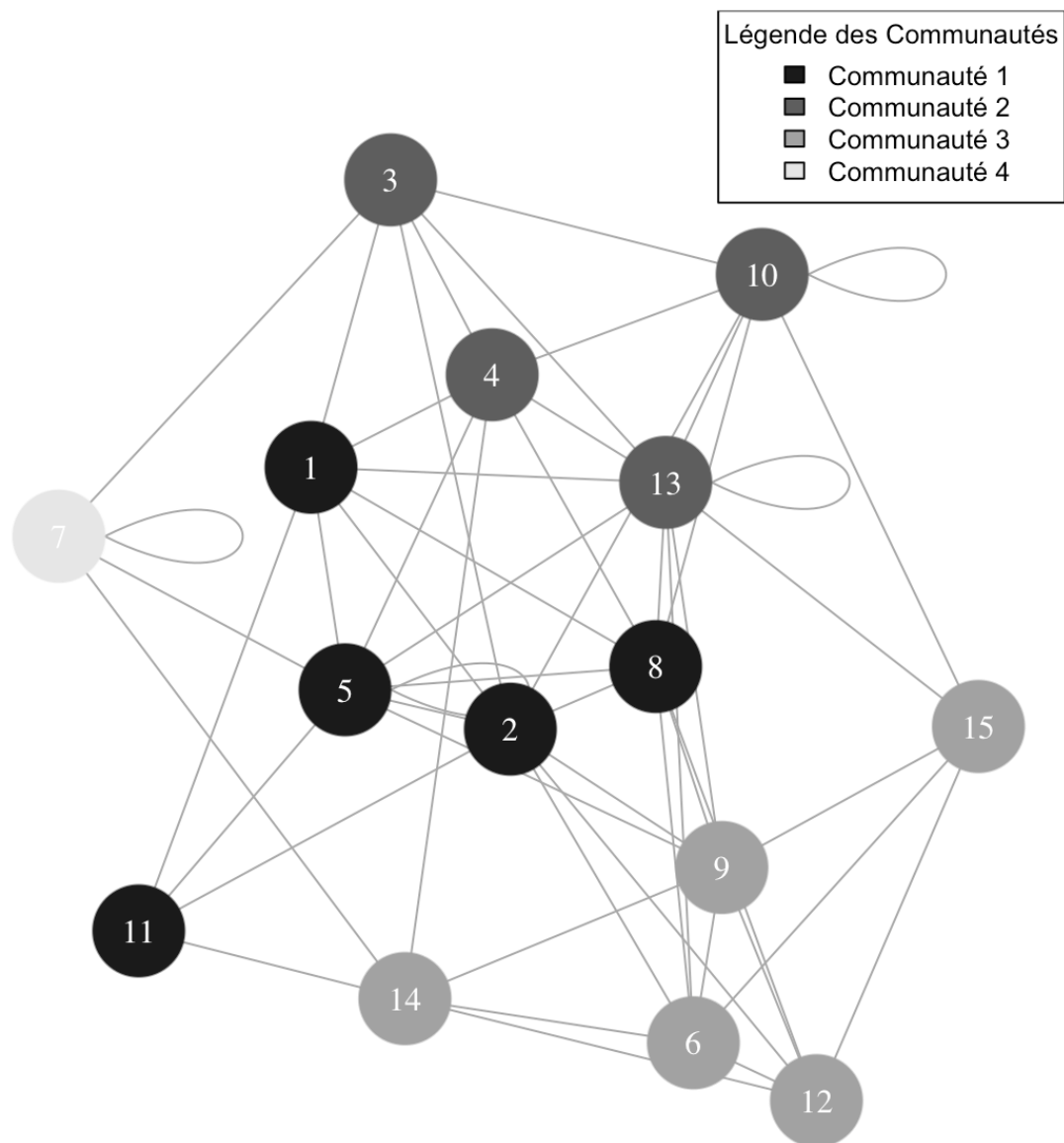


Figure 16: Résultat finale de l'algorithme

3.5 Description algorithmique

Entrée : Graphe simple $G = (V, E)$.

Sortie: K communautés

1. Fonction `detect_communities_louvain(graph)`:

- Appliquer l'algorithme Louvain pour détecter les communautés :
- Calculer la modularité pour partitionner les sommets en communautés
- Retourner l'objet de communautés détectées (communities)

2. Fonction `construct_laplacian_matrix(graph)`:

- Calculer la matrice d'adjacence (A) :
- Pour chaque paire de sommets (i, j) :
- S'il y a une arête entre i et j : - $A[i, j] = 1$
- Sinon : - $A[i, j] = 0$
- Calculer la matrice des degrés (D) :
- Pour chaque sommet i :
- $D[i, i] = \text{degré du sommet } i$ (nombre d'arêtes connectées au sommet i)
- Calculer la matrice Laplacienne (L) :
- $L = D - A$
- Afficher les matrices : A (matrice d'adjacence), D (matrice des degrés) , L (matrice Laplacienne)
- Retourner L

3. Fonction `spectral_clustering (graph, num_clusters)`:

- Construire la matrice Laplacienne (L) en appelant `construct_laplacian_matrix`
- Calculer la décomposition spectrale de L :
- Obtenir les valeurs propres et les vecteurs propres de L
- $L = U * \Lambda * U^T$ (U = vecteurs propres, Λ = valeurs propres)

- Sélectionner les vecteurs propres associés aux num_clusters plus petites valeurs propres : -
Utiliser les premières colonnes de U
- Appliquer k-moyennes sur ces vecteurs propres :
- Entrée : vecteurs propres sélectionnés
- Sortie: clusters
- Retourner les indices des clusters pour chaque sommet

L'algorithme que nous proposons détecte des communautés dans un graphe simple en utilisant deux méthodes principales. Tout d'abord, la fonction `detect_communities_louvain` applique l'algorithme de Louvain, qui regroupe les points en fonction de leurs connexions, cherchant à maximiser les relations internes au sein de chaque groupe. Ensuite, la fonction `construct_laplacian_matrix` crée une matrice qui résume les connexions du graphe, en montrant quelles paires de points sont reliées et combien de connexions chaque point a. Enfin, la fonction `spectral_clustering` utilise cette matrice pour identifier les groupes en appliquant une méthode de regroupement appelée k-moyennes. En gros, cet algorithme combine des techniques de connexion et d'analyse pour identifier des communautés dans le graphe.

3.6 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle approche pour la détection de communautés dans les réseaux, fondée sur les propriétés spectrales de la matrice Laplacienne et sur l'algorithme de Louvain. Nous avons examiné en détail le fonctionnement de l'algorithme de Louvain ainsi que les méthodes de classification et les calculs associés. Chaque étape de notre approche a été illustrée par des exemples pratiques.

En conclusion, notre approche a abouti à l'identification de quatre communautés distinctes. Ces communautés ne peuvent pas être fusionnées davantage, car la modularité ne peut plus être améliorée. Cela indique que notre méthode a atteint un point optimal en termes de détection de communautés selon le critère de modularité.

4 IMPLÉMENTATION ET EXPÉRIMENTATION

Dans ce chapitre, nous procéderons à l'implémentation détaillée de notre algorithme à l'aide du langage de programmation R. Nous commencerons par exposer de manière exhaustive les étapes requises pour coder l'algorithme, en mettant en lumière les aspects techniques et méthodologiques de chaque phase de développement. Cette présentation inclura une documentation rigoureuse pour garantir une transparence totale et permettre une réplique précise des résultats obtenus.

Après avoir implémenté notre algorithme, nous entreprendrons une comparaison des résultats avec ceux générés par une méthode de classification spectrale modifiée, telle que développée par Mazouz, ainsi qu'avec les résultats obtenus par l'algorithme Louvain [6]. Cette analyse comparative nous permettra d'évaluer la performance relative de notre approche en termes de détection de communautés et de qualité des partitions.

En résumé, ce chapitre proposera une analyse approfondie de l'efficacité et de la précision de notre algorithme en le confrontant à des méthodes éprouvées. Cette évaluation critique et détaillée visera à fournir une appréciation complète de ses performances et à mettre en évidence ses avantages et ses limites par rapport aux méthodes établies.

4.1 Implémentation de l'algorithme

De nos jours, avec l'avancée de la Technologie, une multitude d'environnement de travail s'offrent aux chercheurs et développeurs pour les travaux de grandes envergures sur des très longues durées peu importe les domaines de recherches, en ce qui est de ce présent mémoire, le langage de programmation R et son environnement de travail RStudio ont servi de cadre tout au long de la phase d'implémentation du code. Trois packages aux apports incontournables ont été au centre du code, à savoir : `igraph`, `RSpectra` et `Matrix`.

4.1.1 Environnement

RStudio est un environnement de développement intégré (IDE) dédié au langage de programmation R, facilitant la programmation, l'analyse et la visualisation des données. Il offre une interface utilisateur complète qui comprend une console, un éditeur de scripts, un espace pour visualiser les objets en mémoire, et des outils pour la gestion des packages et des fichiers.

Implementation de l'Algorithme

- **Configuration** : Créer un projet dans RStudio pour organiser les fichiers et les scripts.
- **Codage** : Développer l'algorithme en utilisant le Script Editor. Documenter le code pour assurer la clarté et la reproductibilité.
- **Exécution** : Tester le code dans la console et valider chaque partie à l'aide de tests unitaires et de débogage.

Analyse et Visualisation

- **Résultats** : Utiliser les capacités de visualisation de RStudio pour créer des graphiques et des tableaux à partir des résultats expérimentaux.
- **Comparaison** : Comparer les résultats de votre algorithme avec d'autres méthodes établies en utilisant des mesures quantitatives et qualitatives.

RStudio est essentiel pour le développement et l'évaluation de solutions algorithmiques en R, en offrant des outils pour une programmation efficace, une analyse approfondie, et une documentation complète.

4.1.2 Package

- **Igraph**

Igraph est un package pour R conçu pour la manipulation, l'analyse et la visualisation de graphes et réseaux. Il offre des outils complets pour créer des graphes, calculer diverses mesures, détecter des communautés et visualiser des structures complexes.

Fonctionnalités principales:

- **Création de Graphes** : Créer des graphes à partir de matrices d'adjacence, de listes d'arêtes, ou en générant des graphes aléatoires.
- **Manipulation** : Ajouter, supprimer ou modifier des sommets et des arêtes, et ajuster les attributs des graphes.
- **Analyse** : Calculer des mesures de centralité, détecter des communautés avec des algorithmes comme celui développé par Louvain, et appliquer des algorithmes de parcours tels que DFS et BFS.
- **Visualisation** : Personnaliser et visualiser les graphes avec des options de style et de couleur adaptées aux besoins d'analyse.

Le package igraph est essentiel pour les analyses de réseaux en R, facilitant des opérations complexes sur des structures de graphes tout en offrant des outils robustes pour une interprétation visuelle et quantitative des données de réseau.

- **Rspectra**

RSpectra est un package pour R conçu pour le calcul efficace des valeurs propres et des vecteurs propres des matrices. Il est basé sur la bibliothèque Spectra pour des algorithmes de décomposition spectrale optimisés, surtout pour les grandes matrices et les matrices creuses.

Fonctionnalités principales:

- Calcul des valeurs propres et des vecteurs propres :
 - Matrices denses et creuses : Le package supporte à la fois les matrices denses et creuses.
 - Algorithmes optimisés : Utilise des méthodes telles que les algorithmes de Lanczos et Arnoldi pour le calcul efficace des valeurs propres.
- Décomposition spectrale:
 - Valeurs propres et vecteurs propres : Calcule les valeurs propres et les vecteurs propres associés pour des matrices carrées.
 - Tronçonnage spectral : Permet de trouver un sous-ensemble des valeurs propres et vecteurs propres, particulièrement utile pour les grandes matrices.

Le package RSpecra est un outil puissant et optimisé pour le calcul des valeurs propres et des vecteurs propres dans R. Il est particulièrement utile pour les grandes matrices et les matrices creuses, grâce à ses algorithmes efficaces basés sur la bibliothèque Spectra. Ce paquet est essentiel pour les analyses spectrales et la réduction dimensionnelle dans des contextes de données complexes et volumineuses.

- **Matrix**

Le package Matrix pour R est une bibliothèque spécialisée dans la manipulation et l'analyse de matrices. Il permet une gestion efficace des matrices, qu'elles soient denses ou creuses, avec des outils optimisés pour des calculs numériques avancés.

Fonctionnalités principales:

- Création de matrices:
 - Matrices denses : Stocke tous les éléments explicitement.
 - Matrices creuses : Stocke uniquement les éléments non nuls pour une gestion mémoire optimisée.

- Opérations sur les matrices:
 - Opérations arithmétiques, résolutions de systèmes d'équations, inversion de matrices, et calculs de déterminants.
 - Décompositions telles que SVD (Singular Value Decomposition) et LU (LU Decomposition).
- Fonctions spécialisées:
 - Manipulation efficace des matrices creuses.
 - Calculs linéaires avec des fonctions optimisées pour les grandes matrices.

Le paquet **Matrix** est essentiel pour les calculs avec des matrices, offrant des outils puissants pour manipuler et analyser efficacement les données matricielles.

4.2 Mise en œuvre de l'algorithme

4.2.1 Chargement des bibliothèques

Nous commençons par charger les bibliothèques nécessaires :

- **Igraph** pour la manipulation et la visualisation des graphes.
- **RSpectra** pour les calculs de décomposition spectrale.
- **Matrix** pour les opérations sur les matrices.

4.2.2 Fonctions principales

Plot_original_graph : Cette fonction permet de visualiser le graphe avec des étiquettes pour les sommets. Les paramètres incluent la taille des sommets, la taille des étiquettes et la couleur des étiquettes.

L'input est un graphe et l'output est une visualisation du graphe où :

- Les sommets ont une taille de 20.

Chaque sommet est affiché avec son nom en texte noir et de taille 0.8.

- Le graphe a un titre personnalisé.

Cela permet de voir les connexions entre les sommets avec une présentation claire et simple.

```
FONCTION plot_original_graph(graph, title)

// Étape 1 : Tracer le graphe avec les paramètres spécifiés

tracer le graphe avec :

    taille des sommets fixée à 20

    étiquettes des sommets affichées avec les noms des sommets

    taille du texte des étiquettes fixée à 0.8

    couleur du texte des étiquettes fixée à "black"

    titre du graphique fixé par title

FIN FONCTION
```

Detect_communities_louvain : Cette fonction utilise l'algorithme de Louvain pour détecter les communautés dans le graphe.

Input (Entrée):

1.Nœuds: A, B, C, D, E, F, G, H,

2. Tableau des arêtes et poids

Arête	Poids
A-B	5
A-C	3
B-C	4
B-D	2
E-F	6
E-G	5
F-G	4
H-G	7

Sorties:

Communautés détectées

Communauté 1 : {A, C, D}

Communauté 2 : {E, F, G}

Communauté 3: {H}

Entrées : Le graphe est constitué de nœuds et d'arêtes. Les nœuds représentent des entités (comme des amis ou des utilisateurs), tandis que les arêtes représentent les relations entre ces entités, chaque arête ayant un poids qui indique la force de la connexion. Par exemple, une arête avec un poids de 5 signifie une connexion forte, tandis qu'une arête avec un poids de 2 indique une connexion plus faible.

Sorties : Après l'exécution de la fonction `detect_communities_louvain`, les nœuds sont regroupés en communautés basées sur la force de leurs connexions. Les communautés détectées indiquent quels nœuds sont étroitement liés entre eux. Les communautés sont le résultat de l'algorithme de Louvain, qui identifie les groupes de nœuds ayant des interactions plus fortes entre eux par rapport aux autres groupes.

```

FONCTION detect_communities_louvain(graph)
  // Étape 1 : Initialisation
  // Aucune action nécessaire ici.

  // Étape 2 : Préparation du Graphe
  SI graph EST non orienté ET a des poids d'arêtes ALORS
    // Continuer avec l'algorithme Louvain
  SINON
    // Erreur : Le graphe doit être non orienté avec des poids d'arêtes
    RETOURNER "Erreur : Le graphe doit être non orienté avec des poids d'arêtes"
  FIN SI

  // Étape 3 : Détection Initiale des Communautés
  communities <- cluster_louvain(graph)

  // Étape 4 : Retour des Communautés
  RETOURNER communities
FIN FONCTION

```

Fonction construct_laplacian_matrix(graph) : La matrice Laplacienne est essentielle pour la classification spectrale. Elle est calculée comme la différence entre la matrice des degrés et la matrice d'adjacence.

Entrées :

Matrice d'adjacence(A)

	A	B	C	D	E	F	G	H	I	J
A	0	5	3	0	0	0	0	0	0	0
B	5	0	4	2	0	0	0	0	0	0
C	3	4	0	0	0	0	0	0	0	0
D	0	2	0	0	0	0	0	0	0	0
E	0	0	0	0	0	6	5	0	0	0
F	0	0	0	0	6	0	4	0	0	0
G	0	0	0	0	5	4	0	0	0	0
H	0	0	0	0	0	0	0	0	7	0
I	0	0	0	0	0	0	0	7	0	2
J	0	0	0	0	0	0	0	0	2	0

Matrice des degrés (D)

- **Nœud A** : Degré de 2, car il est connecté à B et C.
- **Nœud B** : Degré de 3, car il est connecté à A, C et D.
- **Nœud C** : Degré de 2, car il est connecté à A et B.
- **Nœud D** : Degré de 1, car il est connecté à B.
- **Nœud E** : Degré de 2, car il est connecté à F et G.
- **Nœud F** : Degré de 2, car il est connecté à E et G.
- **Nœud G** : Degré de 2, car il est connecté à E et F.
- **Nœud H** : Degré de 1, car il est connecté à I.
- **Nœud I** : Degré de 3, car il est connecté à H et J.
- **Nœud J** : Degré de 1, car il est connecté à I.

Cette matrice indique le nombre de connexions directes que chaque nœud a avec d'autres nœuds dans le graphe.

Sorties:

Matrice laplacienne (L)

	A	B	C	D	E	F	G	H	I	J
A	2	-5	-3	0	0	0	0	0	0	0
B	-5	3	-4	-2	0	0	0	0	0	0
C	-3	-4	0	0	0	0	0	0	0	0
D	0	-2	0	1	0	0	0	0	0	0
E	0	0	0	0	2	-6	-5	0	0	0
F	0	0	0	0	-6	2	-4	0	0	0
G	0	0	0	0	-5	-4	2	0	0	0
H	0	0	0	0	0	0	0	1	-7	0
I	0	0	0	0	0	0	0	-7	3	-2
J	0	0	0	0	0	0	0	0	-2	1

Étape 1 : Calcul de la matrice d'adjacence (A) : La fonction commence par extraire la matrice d'adjacence du graphe, qui est essentielle pour comprendre comment les nœuds sont reliés.

Étape 2 : Calcul de la matrice des degrés (D) : Ensuite, elle construit la matrice des degrés en fonction des connexions de chaque nœud. Cela permet de déterminer combien de connexions sortantes chaque nœud a.

Sorties :

Matrice laplacienne (L): La matrice laplacienne est ensuite construite en soustrayant la matrice d'adjacence de la matrice des degrés. Cela fournit une représentation des relations structurelles dans le graphe. Les valeurs sur la diagonale représentent les degrés des nœuds, tandis que les valeurs hors diagonale indiquent les relations négatives entre les nœuds.

```

FONCTION construct_laplacian_matrix(graph)
  // Étape 1 : Calcul de la Matrice d'Adjacence (A)
  A <- extraire la matrice d'adjacence du graph
  Afficher "Matrice d'adjacence (A) :"
  Afficher A
  // Étape 2 : Calcul de la Matrice des Degrés (D)
  D <- construire la matrice des degrés avec les degrés des sommets
  Afficher "Matrice des degrés (D) :"
  Afficher D
  // Étape 3 : Construction de la Matrice Laplacienne (L)
  L <- D - A
  Afficher "Matrice Laplacienne (L) :"
  Afficher L
  // Étape 4 : Retourner la Matrice Laplacienne
  RETOURNER L
FIN FONCTION

```

Fonction_spectrale_clustering : La classification spectrale utilise la décomposition spectrale de la matrice Laplacienne pour effectuer un clustering k-moyennes.

Entrées :

La matrice laplacienne (L) obtenue à partir de la matrice d'adjacence et de la matrice des degrés.

Matrice Laplacienne(L) :

	A	B	C	D	E	F	G	H	I	J
A	2	-5	-3	0	0	0	0	0	0	0
B	-5	3	-4	-2	0	0	0	0	0	0
C	-3	-4	0	0	0	0	0	0	0	0
D	0	-2	0	1	0	0	0	0	0	0
E	0	0	0	0	2	-6	-5	0	0	0
F	0	0	0	0	-6	2	-4	0	0	0
G	0	0	0	0	-5	-4	2	0	0	0
H	0	0	0	0	0	0	0	1	-7	0
I	0	0	0	0	0	0	0	-7	3	-2
J	0	0	0	0	0	0	0	0	-2	1

Nous décomposons la matrice Laplacienne en valeurs propres et vecteurs propres. Supposons que nous obtenons les résultats suivants :

-Valeurs propres : $\lambda_1 = 0, \lambda_2 = 1, \lambda_3 = 2, \lambda_4 = 3, \lambda_5 = 3, \lambda_6 = 3, \lambda_7 = 3, \lambda_8 = 4, \lambda_9 = 4, \lambda_{10} = 4$

-Vecteurs propres : $[v_1, v_2, \dots, v_{10}]$

Sélection des vecteurs propres

Entrées:

-num_clusters = 3 (nombre de clusters souhaités).

Nous extrayons les vecteurs propres correspondant aux num_clusters plus petites valeurs propres.

Pour cet exemple, nous choisissons les deux premiers vecteurs propres :

-Vecteurs propres sélectionnés : $v1$ et $v2$

Application de l'algorithme k-moyennes

Nous appliquons l'algorithme k-moyennes sur les vecteurs propres extraits pour réaliser le clustering comme vu dans le chapitre 3.

Entrées:

- Les vecteurs propres sélectionnés ($v1$, $v2$) et $\text{num_clusters} = 3$.

En appliquant l'algorithme k-moyennes, nous obtenons les résultats suivants :

- Cluster 1 : A, B, C, D (ce groupe représente des amis interconnectés)
- Cluster 2 : E, F, G (ce groupe représente un autre ensemble d'amis)
- Cluster 3 : H, I, J (ce groupe représente une communauté distincte)

Sorties :

Les labels de clustering retournés par la fonction pourraient être :

- Cluster 1: A, B, C, D
- Cluster 2: E, F, G
- Cluster 3 : H, I, J

Dans cet exemple complet, la fonction `_spectrale_clustering` a permis de diviser le graphe d'amitié en trois groupes distincts, révélant ainsi la structure de la communauté au sein du réseau.

Les étapes incluent la construction de la matrice laplacienne, la décomposition spectrale, la sélection des vecteurs propres, et enfin l'application de l'algorithme k-moyennes. Les résultats du clustering aident à identifier des groupes d'amis ou d'associés dans un réseau social.

```

FONCTION spectral_clustering (graph, num_clusters)

// Étape 1 : Construction de la Matrice Laplacienne

L <- construire la matrice Laplacienne du graph

// Étape 2 : Décomposition Spectrale

eigen_result <- décomposer L en valeurs propres et vecteurs propres

AFFICHER "Valeurs propres :"

AFFICHER eigen_result$values

AFFICHER "Vecteurs propres :"

AFFICHER eigen_result$vectors

// Étape 3 : Sélection des Vecteurs Propres

eigen_vectors <- extraire les vecteurs propres pour les num_clusters plus petites valeurs propres

AFFICHER "Vecteurs propres pour les num_clusters plus petites valeurs propres :"

AFFICHER eigen_vectors

// Étape 4 : Application de l'Algorithme k-moyennes

clustering <- appliquer k-moyennes sur eigen_vectors avec num_clusters

AFFICHER "Résultats du clustering k-moyennes :"

AFFICHER clustering

// Étape 5 : Retourner les Labels de Clustering

RETOURNER clustering$cluster

FIN FONCTION

```

plot_community_graph: Cette fonction visualise le graphe en coloriant les sommets en fonction des communautés détectées. Elle inclut également une légende pour identifier les différentes communautés.

Entrées :

Graphe

Communautés :

Communauté 1 : {A, C, D}

Communauté 2 : {E, F, G}

Communauté 3 : {H}

Extraction des communautés (à partir de l'input) :

Le graphe contient 3 communautés :

- A, C, D appartiennent à la **communauté 1**.
- E, F, G appartiennent à la **communauté 2**.
- H appartient à la **communauté 3**.

Couleurs pour les communautés :

Trois couleurs distinctes sont générées pour représenter ces groupes :

- **Communauté 1** est colorée en **noir**.
- **Communauté 2** est colorée en **gris**.
- **Communauté 3** est colorée en **blanc**.

Application des couleurs aux sommets :

Les nœuds A, C, D sont colorés en bleu, E, F, G en rouge, et H en vert pour visualiser facilement les différents groupes.

Contours des sommets :

Des contours translucides sont ajoutés autour des sommets pour un meilleur contraste, ce qui rend les nœuds plus visibles, surtout dans des graphes denses.

Création d'une légende :

Une légende est ajoutée pour montrer que :

- **Noir** correspond à la communauté 1 (A, C, D).
- **Gris** correspond à la communauté 2 (E, F, G).
- **Blanc** correspond à la communauté 3 (H).

Visualisation du graphe :

Le graphe est tracé avec des nœuds colorés selon leur communauté et des liens (arêtes) en gris, avec le titre “Visualisation des communautés”.

Ajout de la légende :

La légende apparaît en haut à droite pour aider à identifier les couleurs des différentes communautés.

Sorties :

Un **graphe visuel** montrant :

- A, C, D en **noir** (Communauté 1),
- E, F, G en **gris** (Communauté 2),
- H en **blanc** (Communauté 3),
- Des **arêtes grises** reliant les sommets selon leurs connexions,
- Une **légende** expliquant la correspondance entre couleurs et communautés.

En résumé, l'**entrée** est un graphe avec ses communautés et un titre, et la sortie est une visualisation graphique avec des nœuds colorés, des arêtes grises, et une légende expliquant les couleurs des communautés.

```

FONCTION plot_community_graph(graph, communities, title)
// Étape 1 : Extraction des membres des communautés
membership <- obtenir les membres des communautés

// Étape 2 : Définition des couleurs pour les communautés
num_communities <- nombre de communautés uniques dans membership
colors <- générer une palette de couleurs avec num_communities couleurs

// Étape 3 : Application des couleurs aux sommets en fonction de leur communauté
vertex_colors <- assigner les couleurs à chaque sommet en fonction de membership

// Étape 4 : Définition des couleurs des contours pour le contraste
border_colors <- ajuster la couleur des contours en réduisant l'opacité

// Étape 5 : Création d'une légende
legend_labels <- créer les étiquettes de légende avec "Communauté" suivi du numéro
legend_colors <- utiliser les couleurs définies pour la légende

// Étape 6 : Visualisation du graphe avec couleurs
tracer le graphe avec :
    taille des sommets fixée à 20
    couleur des sommets définie par vertex_colors
    couleur du contour des sommets définie par border_color
    étiquettes des sommets affichées avec le nom du sommet
    couleur du texte des étiquettes fixée à "black"
    couleur des arêtes fixée à "grey"
    titre du graphe fixé par title

// Étape 7 : Ajouter la légende
ajouter une légende en haut à droite du graphe avec :
    étiquettes définies par legend_labels
    couleurs de la légende définies par legend_colors
    titre de la légende fixé à "Légende des Communautés"
    taille du texte de la légende fixée à 0.8
FIN FONCTION

```

4.3 Exécution de l'algorithme

Étape 1 : Génération du Graphe

Nous générons une matrice d'adjacence aléatoire pour créer un graphe non dirigé. Cette matrice est constituée de valeurs binaires (0 ou 1), où chaque élément représente la présence ou l'absence d'une connexion entre deux sommets du graphe. Pour garantir que le graphe est non dirigé, nous symétrisons la matrice en ajoutant sa transposée et en normalisant les valeurs. Les éléments diagonaux sont ensuite mis à zéro pour éliminer les boucles auto-référentielles. À partir de cette matrice d'adjacence, nous créons un graphe.

Étape 2 : Visualisation Initiale

Nous affichons le graphe d'origine pour obtenir une vue d'ensemble de sa structure. Cette visualisation permet de voir les connexions entre les sommets et d'obtenir une représentation visuelle du réseau.

Étape 3 : Calcul de la matrice laplacienne

Nous calculons la matrice Laplacienne du graphe, qui est une matrice clé utilisée dans l'analyse des graphes. La matrice Laplacienne est définie comme la différence entre la matrice de degré (une matrice diagonale des degrés des sommets) et la matrice d'adjacence du graphe. Cette matrice est ensuite affichée pour une analyse plus approfondie.

Étape 4 : Détection des communautés

Nous détectons les communautés dans le graphe en utilisant l'algorithme de Louvain. Cet algorithme permet d'identifier des groupes de sommets qui sont densément connectés entre eux tout en ayant moins de connexions avec le reste du graphe. Après l'application de l'algorithme, nous affichons le nombre de communautés détectées, ce qui donne un aperçu de la structure du graphe en termes de groupes cohérents.

Étape 5 : Classification spectrale

Nous effectuons une classification spectrale pour partitionner le graphe en un nombre spécifié de clusters. Cette méthode repose sur les vecteurs propres de la matrice Laplacienne pour déterminer les groupes de sommets. Les résultats de la classification spectrale sont ensuite affichés, montrant à quel cluster chaque sommet appartient.

Étape 6 : Visualisation des communautés

Enfin, nous visualisons le graphe en utilisant une coloration des communautés détectées par l'algorithme de Louvain. Cette visualisation colorée permet de voir clairement les différentes communautés et la manière dont elles sont regroupées dans le graphe. Cela aide à interpréter les résultats de la détection des communautés et à observer la structure du réseau en termes de groupes fonctionnels.

Pseudo Code :

Début

Étape 1 : Génération de la Matrice d'Adjacence

Définir num_vertices

Fixer la graine aléatoire (set.seed(123))

Générer une matrice aléatoire custom_adj_matrix de dimensions num_vertices x num_vertices avec des valeurs 0 et 1, avec une probabilité de 0.3 pour les 1

Symétriser la matrice en la rendant égale à la somme de la matrice et de sa transposée, divisée par 2

Mettre les éléments diagonaux de la matrice à 0

Étape 2 : Création du Graphe et Affichage du Graphe

Créer un graphe graph_from_adj_matrix à partir de custom_adj_matrix

Attribuer des labels numériques aux sommets du graphe

Afficher le graphe avec les étiquettes des sommets

Étape 4 : Calcul et Affichage de la Matrice Laplacienne

Calculer la matrice Laplacienne Laplacian_matrix du graphe

Afficher la matrice Laplacienne

Étape 5 : Détection des Communautés avec l'Algorithme de Louvain

Détecter les communautés communities_louvain avec l'algorithme de Louvain

Compter le nombre de communautés distinctes

Afficher le nombre de communautés détectées

Étape 6 : Classification Spectrale

Définir le nombre de clusters num_clusters (ex: 3)

Effectuer une classification spectrale spectral_result sur le graphe avec num_clusters

Afficher les résultats de la classification spectrale

Fin

Cette présentation détaillée couvre toutes les étapes nécessaires pour implémenter et évaluer un algorithme de détection de communautés dans un graphe en R. Chaque fonction est expliquée en termes de son rôle et de son utilisation, et les processus sont illustrés avec des exemples de code pour une meilleure compréhension.

4.4 Évaluation comparative de la modularité sous variation du paramètre de mixage

Cette étude vise à générer plusieurs graphes en faisant varier le paramètre de mixage μ afin d'évaluer et comparer l'efficacité de différents algorithmes de détection de communautés. Nous calculerons la modularité pour analyser les résultats obtenus.

Trois algorithmes seront étudiés :

- **Classification avec Louvain (CL)** : Une version optimisée de l'algorithme Louvain, développée pour cette étude.
- **Algorithme Louvain (AL)** : L'algorithme standard, utilisé comme référence.
- **Classification spectrale de Mazzouz (CB)** : Une méthode basée sur la décomposition spectrale des graphes.

Cette approche permettra d'examiner la robustesse des algorithmes dans des conditions variées. En évaluant la modularité, nous mesurerons la qualité des partitions. Des visualisations seront également générées pour faciliter la comparaison visuelle des partitions obtenues et mieux comprendre les différences en termes de structure et de répartition des communautés [10].

4.4.1 Paramètre de mixage

Le paramètre de mixage μ reflète la tendance des nœuds d'une communauté à interagir principalement au sein de leur propre groupe, par rapport à ceux d'autres communautés. Des valeurs proches de 1 indiquent une forte cohésion interne, tandis que des valeurs proches de 0 révèlent plus d'interactions avec des nœuds externes [35].

Ce paramètre est essentiel pour évaluer la cohésion et la robustesse des communautés, bien qu'il doive être interprété en complément de la modularité ou de la densité. Lorsque la valeur de μ est proche de 1, cela indique que les nœuds de la communauté ont une forte cohésion interne, signifiant qu'ils privilégient les interactions avec leurs pairs au sein du même groupe. À l'inverse, une valeur

proche de 0 suggère que les nœuds interagissent davantage avec des nœuds d'autres communautés, ce qui peut indiquer une intégration ou une interconnexion plus forte avec le réseau global.

Ce paramètre est essentiel pour évaluer non seulement la cohésion d'une communauté, mais aussi sa robustesse face à des perturbations. En effet, une communauté avec une forte cohésion interne est souvent plus résiliente. Cependant, pour obtenir une image complète de la structure du réseau, il est crucial d'interpréter le paramètre de mixage en complément d'autres mesures telles que la modularité, qui quantifie la qualité de la division en communautés, ou la densité, qui mesure le nombre de liens présents par rapport au nombre maximal possible de liens dans un réseau. Ensemble, ces métriques permettent d'analyser en profondeur la structure et le comportement des communautés au sein d'un réseau.

4.4.2 Notion de modularité

La modularité est une mesure fondamentale en théorie des réseaux et en analyse des données qui permet d'évaluer la qualité d'une partition d'un réseau en communautés. Développée par des chercheurs comme [Fortunato, S. \(2010\)](#) [16], elle est largement utilisée dans le domaine de la détection de communautés.

La modularité quantifie la différence entre le nombre d'arêtes qui existent réellement au sein des communautés et le nombre d'arêtes qui seraient attendues dans un réseau aléatoire [34].

Elle mesure ainsi le degré de séparation des communautés : plus les nœuds d'une communauté sont interconnectés entre eux et moins ils interagissent avec d'autres communautés, plus la modularité est élevée [26].

Une modularité:

Proche de 1 : Indique une bonne séparation entre les communautés, suggérant que les nœuds à l'intérieur d'une communauté sont fortement connectés entre eux [41].

Autour de 0 : Suggère que la partition ne montre pas de structure significative, et que les communautés détectées ressemblent à un agencement aléatoire [37].

Négative : Indique que les nœuds sont plus interconnectés avec d'autres communautés qu'au sein de leur propre communauté [35].

L'algorithme demande un certain nombre de paramètres tels que :

- n : le nombre de nœuds
- μ : le paramètre de mixage.

- **Test pour $n=100$**

n= 100	Modularité		
	CL	AL	CB
$\mu =0.0$	0.7	0.85	0.7
$\mu =0.1$	0.66	0.74	0.65
$\mu =0.2$	0.55	0.72	0.6
$\mu =0.3$	0.53	0.7	0.54
$\mu =0.4$	0.49	0.67	0.5
$\mu =0.5$	0.45	0.61	0.43
$\mu =0.6$	0.32	0.55	0.36
$\mu =0.7$	0.31	0.47	0.29
$\mu =0.8$	0.3	0.43	0.27
$\mu =0.9$	0.26	0.32	0.261
$\mu =1$	0.24	0.3	0.26

Tableau 5: Résultats de la modularité pour le graphe de 100 nœuds en variant le paramètre de mixage μ

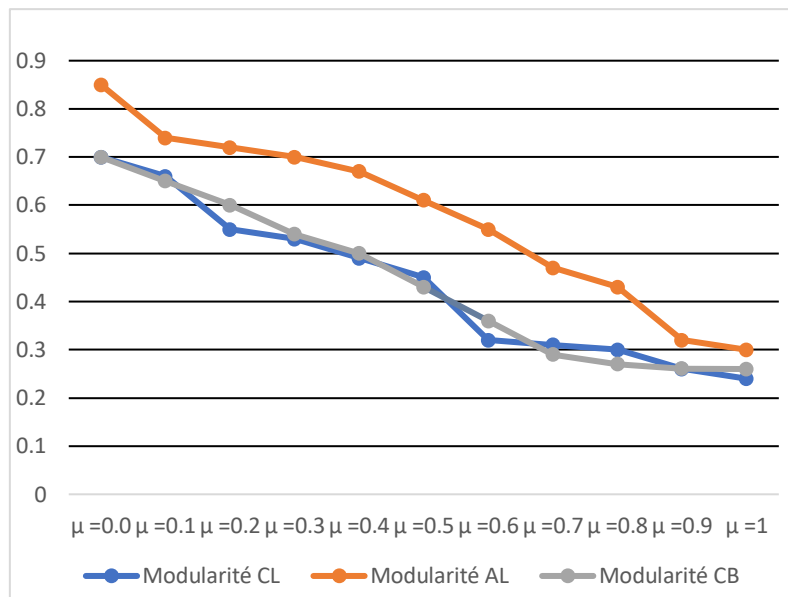


Figure 17: Variation de la modularité en fonction du paramètre de mixage μ pour le graphe de 100 nœuds

Pour $n=100$, nos résultats de modularité sont compétitifs par rapport à ceux de l'algorithme CB. En particulier, pour un paramètre de mixage compris entre 0,7 et 0,8, nos résultats surpassent ceux de l'algorithme CB. Cependant, nous remarquons également que l'algorithme AL reste nettement supérieur aux deux autres, affichant des performances de modularité toujours meilleures, quelle que soit la valeur de μ .

- **Test pour $n=1000$**

n= 1000	Modularité		
	CL	AL	CB
$\mu =0.0$	0.91	0.94	0.93
$\mu =0.1$	0.81	0.88	0.88
$\mu =0.2$	0.8	0.83	0.83
$\mu =0.3$	0.79	0.79	0.78
$\mu =0.4$	0.73	0.78	0.73
$\mu =0.5$	0.73	0.73	0.65
$\mu =0.6$	0.6	0.63	0.61
$\mu =0.7$	0.56	0.6	0.55
$\mu =0.8$	0.47	0.57	0.43
$\mu =0.9$	0.41	0.56	0.38
$\mu =1$	0.32	0.4	0.22

Tableau 6: Résultats de la modularité pour le graphe de 1000 nœuds en variant le paramètre de mixage μ

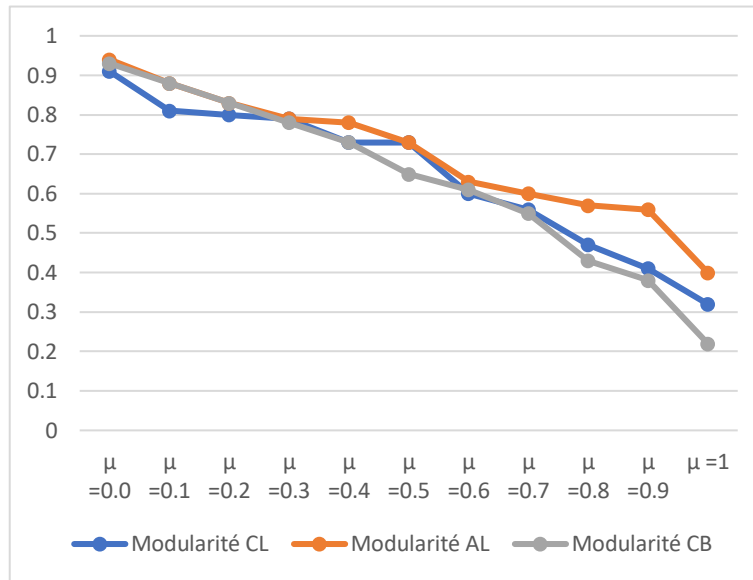


Figure 18: Variation de la modularité en fonction du paramètre de mixage μ pour le graphe de 1000 noeuds

Pour $n=1000$, nos résultats de modularité sont moins bons par rapport aux algorithmes AL et CB. Cependant, nous remarquons qu'après un paramètre de mixage supérieur à 0,4, nos résultats surpassent ceux de l'algorithme CB. En outre, les résultats de la modularité de l'algorithme AL demeurent toujours les meilleurs, quel que soit la valeur de μ utilisée. Cela indique que, même si notre algorithme présente des performances inférieures à AL et CB pour les valeurs plus faibles de μ , il peut rivaliser avec CB à des valeurs plus élevées de ce paramètre.

3. Test pour $n=10000$

n= 10000	Modularité		
	CL	AL	CB
$\mu = 0.0$	0.92	0.93	0.85
$\mu = 0.1$	0.85	0.92	0.74
$\mu = 0.2$	0.74	0.87	0.67
$\mu = 0.3$	0.67	0.74	0.61
$\mu = 0.4$	0.61	0.72	0.55
$\mu = 0.5$	0.55	0.64	0.51
$\mu = 0.6$	0.54	0.61	0.47
$\mu = 0.7$	0.5	0.58	0.42
$\mu = 0.8$	0.48	0.51	0.38
$\mu = 0.9$	0.4	0.5	0.33
$\mu = 1$	0.39	0.48	0.28

Tableau 7: Résultats de la modularité pour le graphe de 10000 nœuds en variant le paramètre de mixage μ

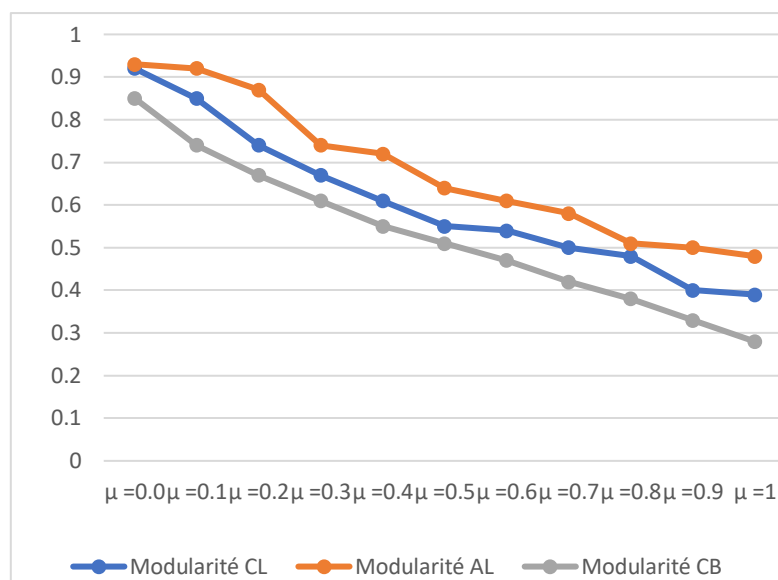


Figure 19: Variation de la modularité en fonction du paramètre de mixage μ pour le graphe de 10000 nœuds

Nous avons constaté que pour $n=10000$, nos résultats de modularité sont meilleurs que ceux de l'algorithme CB et comparables à ceux de l'algorithme AL, même si AL reste le meilleur. Lorsque le paramètre de mixage est faible, les résultats de modularité sont vraiment excellents, mais ils se dégradent à mesure que μ augmente. Cette dégradation indique que notre méthode est efficace dans des situations où les communautés sont bien séparées, mais qu'elle montre une sensibilité accrue lorsque la complexité du réseau augmente. Les performances de la modularité révèlent des résultats intéressants selon les valeurs choisies pour le paramètre de mixage, soulignant l'impact de celui-ci sur l'efficacité de la détection des communautés.

Interprétation des résultats

L'interprétation des résultats souligne les performances de notre algorithme en matière de modularité par rapport aux algorithmes AL et CB. Globalement, notre méthode montre une modularité compétitive, dépassant celle de CB dans certaines conditions et s'approchant des résultats d'AL, qui demeure le meilleur dans tous les cas. Les variations observées en fonction de la taille du réseau et du paramètre de mixage montrent que la modularité est fortement influencée par ces facteurs. Ces résultats indiquent également que notre algorithme présente une capacité d'adaptation aux niveaux de complexité croissants des réseaux. En outre, les performances en matière de modularité soulignent l'importance de la conception algorithmique pour identifier efficacement des structures communautaires significatives, tout en suggérant des pistes d'amélioration pour optimiser notre approche dans des contextes variés.

4.5 Visualisation des différents graphes finaux

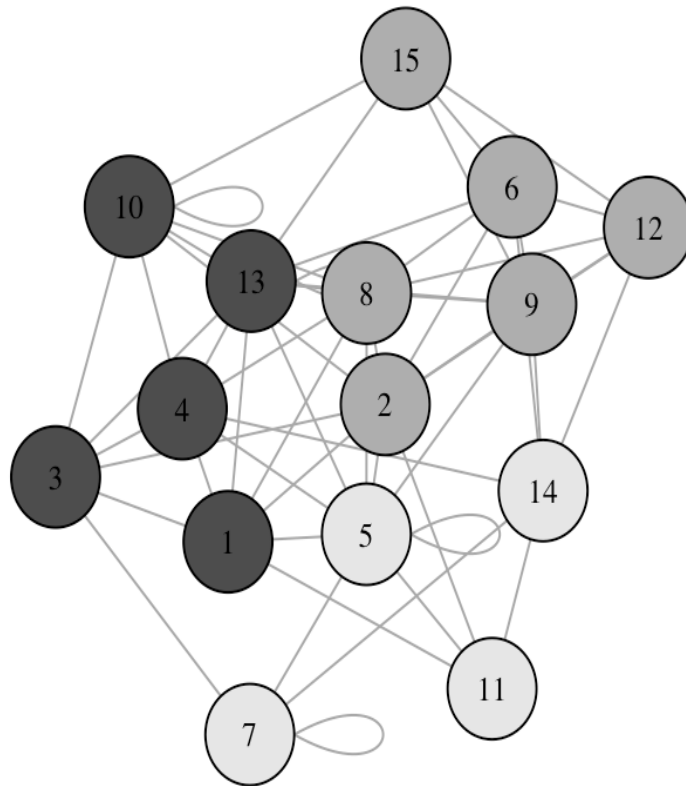


Figure 20: Graphe final détecté par l'algorithme Louvain

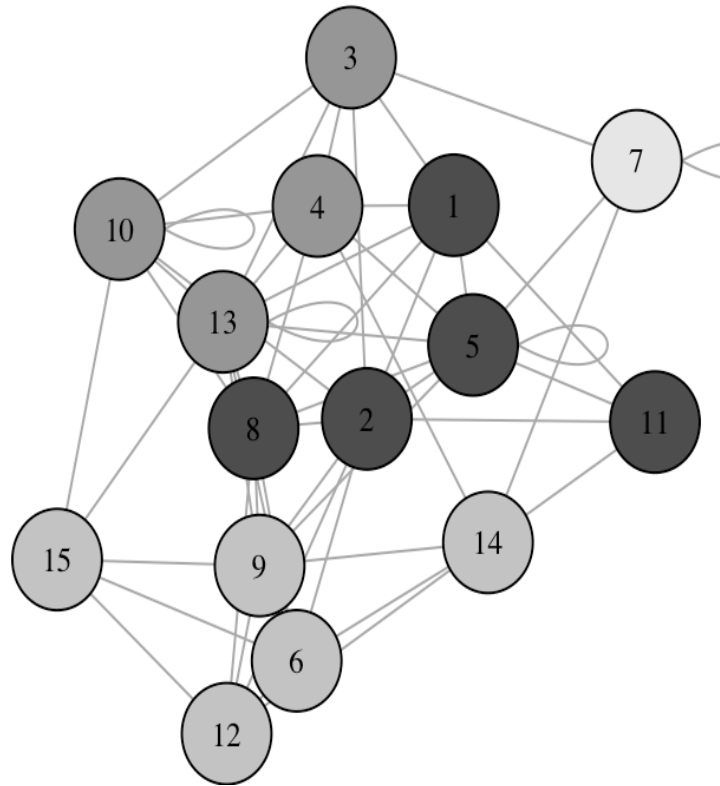


Figure 21: Graphe final détecté par l'algorithme Mazzouz

En comparant les résultats des trois algorithmes de détection de communautés plusieurs observations clés se dégagent. Notre méthode et celle de Mazzouz détectent chacune quatre communautés, et ces communautés sont identiques dans les deux méthodes, ce qui suggère une cohérence dans leurs critères de détection. En revanche, l'algorithme Louvain identifie seulement trois communautés, indiquant une approche différente qui favorise des regroupements plus larges pour maximiser la modularité.

Cette différence révèle que Louvain pourrait privilégier une vue plus globalisée du réseau, en fusionnant des communautés plus petites en ensembles plus grands. En comparaison, notre méthode hybride et la méthode de Mazzouz semblent offrir une vue plus détaillée, détectant des structures communautaires plus fines.

Le choix entre ces algorithmes dépendra des objectifs spécifiques de l'analyse. Si l'objectif est d'obtenir une vue plus agrégée et simplifiée du réseau, Louvain pourrait être préférable. En revanche, pour une détection plus précise des structures communautaires, notre approche hybride et celui de Mazzouz pourraient être plus adaptées. Il est également crucial de valider les résultats obtenus en utilisant des métriques supplémentaires et en testant les algorithmes sur divers jeux de données pour assurer leur robustesse et la qualité des communautés détectées.

4.6 Conclusion

Cette section présente en détail notre algorithme de détection de communautés et fournit une évaluation approfondie de ses performances. Nous comparons notre approche avec des méthodes établies, telles que l'Algorithme Louvain (AL), et Classification Spectrale de Mazzouz(CB).

Les résultats expérimentaux permettent d'évaluer la performance de notre algorithme en comparaison avec ces méthodes, d'identifier ses points forts et ses limites, ainsi que de comprendre comment il se positionne par rapport aux autres approches.

5 CONCLUSION GÉNÉRALE ET PERSPECTIVES

5.1 Conclusion générale

Dans ce mémoire, nous avons exploré une approche hybride pour la détection de communautés dans les graphes, combinant deux des méthodes les plus couramment utilisées en théorie des graphes : l'optimisation de la modularité et la classification spectrale. Notre objectif était de tirer parti des avantages complémentaires de ces techniques pour améliorer la précision et l'efficacité des méthodes de détection de communautés.

Nous avons commencé par une revue exhaustive de la littérature sur les graphes, afin de fournir un cadre théorique solide. Cette revue a couvert différents types de graphes, les concepts clés en théorie des graphes, ainsi que les méthodes classiques et modernes de détection de communautés. Cette base théorique a été essentielle pour comprendre les avantages et les limites des diverses approches existantes.

Dans le troisième chapitre, nous avons introduit notre algorithme hybride, qui combine les techniques d'optimisation de la modularité et de classification spectrale. Nous avons détaillé la manière dont l'algorithme utilise ces méthodes pour identifier des communautés dans un réseau, en cherchant à surmonter les limitations observées dans les approches purement basées sur la modularité ou la classification spectrale.

Le chapitre suivant a été dédié à l'implémentation pratique de notre algorithme et à son évaluation par rapport à d'autres méthodes de détection de communautés, notamment l'algorithme Louvain et les méthodes de classification spectrale basées sur les matrices Laplaciennes. Nous avons évalué les performances de notre approche en termes de qualité des partitions communautaires et de temps d'exécution.

Bien que notre approche hybride ait été conçue pour combiner les points forts des méthodes d'optimisation et de classification spectrale, les résultats ont montré qu'elle n'a pas surpassé les performances des algorithmes existants, notamment en termes de qualité de détection des communautés et de performance computationnelle.

La combinaison des techniques d'optimisation et de classification spectrale a peut-être introduit des complexités supplémentaires qui ont impacté l'efficacité globale de l'algorithme. Les paramètres de l'algorithme et les heuristiques utilisées pourraient nécessiter des ajustements pour améliorer la précision.

Il est possible que la méthode hybride n'ait pas réussi à équilibrer efficacement les objectifs d'optimisation de la modularité et les contraintes imposées par la classification spectrale. Des améliorations pourraient inclure une meilleure intégration des deux méthodes ou l'exploration de techniques supplémentaires pour affiner les résultats.

En conclusion, bien que notre approche hybride ait introduit une méthode innovante pour la détection de communautés, les résultats montrent qu'il reste des défis à relever pour atteindre une performance comparable ou supérieure à celle des méthodes existantes. Les perspectives de recherche offrent des directions prometteuses pour affiner l'algorithme et explorer de nouvelles avenues pour améliorer la détection de communautés dans les graphes.

5.2 Perspectives

Dans le cadre de notre travail sur la détection des communautés, plusieurs perspectives d'amélioration ont été identifiées pour optimiser l'algorithme utilisé. D'abord, il est crucial d'explorer et d'ajuster les paramètres et heuristiques de l'algorithme afin d'améliorer ses performances. Une optimisation fine de l'équilibre entre les techniques d'optimisation et de classification spectrale pourrait non seulement améliorer la qualité des partitions communautaires, mais aussi maintenir une efficacité computationnelle élevée.

Ensuite, la validation de notre approche sur une gamme plus large de réseaux et de jeux de données est nécessaire pour mieux comprendre ses forces et ses faiblesses dans divers contextes. Tester l'algorithme dans différents environnements permettrait de comparer ses performances et de réaliser des ajustements en fonction des résultats obtenus.

Par ailleurs, il serait bénéfique de développer de nouvelles méthodes hybrides en combinant les techniques de détection de communautés avec des approches modernes telles que les algorithmes d'apprentissage automatique ou les réseaux de neurones. Ces combinaisons pourraient offrir des solutions plus robustes et significativement améliorer les résultats de la détection.

Enfin, pour rendre notre algorithme plus efficace et applicable à des réseaux de grande taille, il est essentiel de travailler sur la réduction de la complexité computationnelle et l'amélioration de sa scalabilité. En optimisant le temps de calcul et les besoins en ressources, notre algorithme pourrait traiter des réseaux plus volumineux et complexes, répondant ainsi aux exigences croissantes en matière de traitement de données.

Ces perspectives visent à renforcer la robustesse et l'efficacité de notre travail tout en explorant de nouvelles avenues pour des améliorations futures.

Bibliographie

- [1] Aghabozorgi, S., & Ying, H. (2014). Time-series clustering – A decade review. *Information systems*, 53, 16-38.
- [2] Airoldi, E. M., Blei, D. M., Fienberg, S. E., & X. L. (2008). Mixed membership stochastic blockmodels. *Proceedings of the National Academy of Sciences*, 105(50), 20297-20302.
- [3] Arenas, A., Danon, L., Díaz-Guilera, A., Kivela, M., & Latora, V. (2004). Community analysis in social networks. *European Physical Journal B*, 38(2), 373–380.
- [4] Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439), 509–512.
- [5] Berger, H. (2019). Recherche de communautés dans les graphes.
- [6] 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*.
- [7] Borgatti, S. P., & Halgin, D. S. (2011). Analyzing affiliation networks and bipartite graphs. In J. Scott & P. J. Carrington (Eds.), *The SAGE Handbook of Social Network Analysis*.
- [8] Brandes, U., & Erlebach, T. (2005). Network analysis: methodological foundation.
- [9] Brun, C., Herrmann, C., & Guénoche, A. (2004). Clustering proteins from interaction networks for the prediction of cellular functions.
- [10] Chen, C., & Zhang, Y. (2015). A survey on community detection with deep learning. *IEEE Access*, 7, 39801-39819.
- [11] Chen, W., & Zhang, X. (2019). Community detection with enhanced modularity density. *Knowledge-Based Systems*, 162, 438–449.
- [12] Cheng, J., & Li, J. (2013). A comparative evaluation of community detection algorithms. *Knowledge and Information Systems*, 36(2), 391–408.
- [13] Clauset, A. (2005). Finding local community structure in networks. *Physical Review E*, 72(2).
- [14] Dong, Z., Zhang, X., & Luo, B. (2015). An efficient algorithm for overlapping community detection. *Physical A: Statistical Mechanics and its Applications*, 419, 575-585.

- [15] Dong, Z., Zhang, X., Sun, D., & Luo, B. (2016). Overlapping community detection based on network decomposition. *Scientific Reports*, 6(24115).
- [16] Fortunato, S. (2010). Community detection in graphs. *Physics Reports*.
- [17] Fronczak, A., & Fronczak, P. (2010). Community structure in networks: A comparative analysis of methods. *Physica A: Statistical Mechanics and its Applications*, 389(19), 4242–4256.
- [18] Goder, A., & Filkov, V. (2008). Consensus clustering algorithms: comparison and refinement. *Proceedings of the Meeting on Algorithm Engineering and Experiments*, 109–117.
- [19] Govaert, G. (2003). Analyse des données (Traité IC2, série traitement du signal et de l'image).
- [20] Govan, B., & Snape, T. (2009). *Advanced algorithms for network analysis*. Springer.
- [21] Greene, D., & Cunningham, P. Producing a unified graph representation from multiple social network views.
- [22] Guimera, R., Sales-Pardo, M., & Nunes Amaral, L. A. (2007). Module identification in bipartite networks. *Physical Review E*, 76(3), 036102.
- [23] Helene Berger (2019) Recherche de communauté dans les graphes
- [24] Inconnu (2018). Tout est graphe ! Détection de communautés : théorie et retour d'expérience.
- [25] Karrer, B., & Newman, M. E. J. (2009). Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1), 016107.
- [26] Kempe, D., & Agarwal, G. Modularity-maximizing network communities via mathematical programming. *The European Physical Journal B*.
- [27] Koren, T., & Koren, Y. (2007). Fast modularity optimization with vertex mover. In *Proceedings of the SIAM International Conference on Data Mining*.
- [28] Latapy, M., Pons, P., & Bouche, D. (2008). A constant-time algorithm for computing the modularity of a partition of a graph. In *Proceedings of the 6th International Conference on Algorithms and Complexity*.
- [29] Liu, S., & Zhang, Y. (2012). A hybrid approach to community detection in large-scale networks. *IEEE Transactions on Knowledge and Data Engineering*, 24(8), 1434–1447.
- [30] Liu, X., & Murata, T. (2010). An efficient algorithm for optimizing bipartite modularity in bipartite networks. *Journal of Advanced Computational Intelligence and Intelligent Informatics*.

- [31] Mazzouz, B. (2021). Travail exploratrice sur une méthode de détection de communauté dans les réseaux.
- [32] Nathalie Villa-Vialaneix, Taoufiq Dkaki, Sébastien Gadat, Jean-Michel Inglebert, Quoc-Dinh Truong (2011). Recherche et représentation de communautés dans un grand graphe.
- [33] Newman, M. E. J. (2004). Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6), 066133.
- [34] Newman, M. E. J. (2006). Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*.
- [35] Nguyen, T., & Tran, T. (2018). Improving modularity optimization algorithms. *Journal of Computational Mathematics*, 36(2), 215-227.
- [36] Orman, G., & Labatut, V. A comparison of community detection algorithms on artificial à networks. In *Discovery Science*.
- [37] Papadopoulos, F., & Makse, H. A. (2011). Community detection algorithms: A comparative evaluation. *Data Mining and Knowledge Discovery*, 23(3), 421–441.
- [38] Peixoto, T. P. (2014). Hierarchical stochastic block models: Predictive posterior checks and model selection. *Physical Review E*, 89(1), 012804.
- [39] Pons, P., & Latapy, M. (2005). Computing communities in large networks using random walks. *arXiv*.
- [40] Radicchi, F., Castellano, C., Cecconi, F., Loreto, V., & Parisi, D. (2004). Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*.
- [41] Raghavan, U. N., Albert, R., & Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76, 036106.
- [42] Rosvall, M., & Bergstrom, C. T. (2008). Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4), 1118-1123.
- [43] Schaeffer, S. E. (2007). Graph clustering. *Computer Science Review*, 1(1), 27–64.
- [44] Schuetz, P., & Caflisch, A. (2008). Efficient modularity optimization by multistep greedy algorithm and vertex mover refinement.
- [45] Topchy, A., Jain, A. K., & Punch, W. F. (2006). Clustering ensembles: Models of consensus and weak partitions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- [46] Villa-Vialaneix, N., Dkaki, T., Gadat, S., Inglebert, J.-M., & Truong, Q.-D. (2011). Recherche et représentation de communautés dans un grand graphe.

- [47] Xu, Y., & Chen, L. (2014). Modularity density for evaluating community structure in bipartite networks.
- [48] Zhang, Y., & Chen, J. (2017). Community detection and evolution in dynamic networks. *IEEE Transactions on Knowledge and Data Engineering*, 29(2), 238–251.
- [49] Zhou, X., & Wang, W. (2020).. Community detection algorithms in complex networks: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 31(2), 377-394.