REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MOULOUD MAMMERI TIZI OUZOU

FACULTE DE GENIE ELECTRIQUE ET INFORMATIQUE
DEPARTEMENT D'INFORMATIQUE

Mémoire de fin d'études

En vue de l'obtention du diplôme de Master en informatique

Spécialité: Systèmes informatiques

THÈME

Détection de communautés chevauchantes par filtrage des nœuds d'une partition de liens d'un réseau biparti

Dirigé et proposé par :

M^me^. **AÏT EL HADJ Fatiha**

Réalisé et présenté par :

M^elle^. **Hadid Siham**

Promotion 2012

Remerciements

Je tiens d'abord à remercier tout particulièrement ma promotrice, M^{me}. AÏT EL HADJ, pour sa disponibilité et la qualité de son encadrement.

Je remercies également les membres du jury, devant qui j'ai l'honneur d'exposer mon travail, et qui ont pris la peine de lire avec soin, ce mémoire pour juger son contenu.

Mes remerciements, vont aussi à tous les professeurs du département d'informatique de l'université MOULOUD MAMMERI.

Dédicaces

Je dédie ce modeste travail à toute ma famille, qui m'a été la source d'encouragement et d'assurances durant mes séjours, en particulier a :

Mes chers parents,

Mes frères « HAKIM» et « BELkACEM» ,

Mes grands parents « MOULOUD » et « FATMA »,

Toute ma famille,

Toutes mes amies,

A tous ceux qui m'aiment.

Table des matières

Liste des figures

Introduction générale

De nous jours, les réseaux réels sont devenus de plus en plus grands et complexes. De récentes avancées dans le domaine des systèmes complexes ont fait ressortir le rôle central que jouent les graphes dans de nombreux phénomènes. Ces grands graphes, permettent de modéliser les interactions entre les différents acteurs de ces phénomènes complexes, qui interviennent dans de très nombreux domaines : sociologie, biologie, linguistique, physique, informatique, etc.

En effet, il est difficile de travailler sur des réseaux réels car ceux-ci sont difficiles à gérer et coûteux à construire, et de plus on ne peut pas contrôler leurs propriétés, pour cela les chercheurs ont pensé à découper les réseaux en un ensemble de parties dites « communautés », c'est ce qu'on appelle« la détection de communauté dans les réseaux». C'est une problématique assez ancienne, qui tire ses origines du problème de partitionnement de graphe qui est un problème classique en théorie des graphes.

La détection de communauté dans les réseaux joue un rôle important dans l'organisation ou la structuration des réseaux, non seulement en informatique, mais aussi par ses applications dans de nombreux domaines scientifiques. Cependant, la découpe en communautés est un problème complexe (NP-complet), dont la solution ne peut pas être trouvée au moyen d'une méthode de résolution exacte. Dans ce contexte, de nombreuses méthodes ont été développées. En particulier ces méthodes évaluent la qualité d'une partition en maximisent un certain critère. Parmi ces nombreux critères, le critère de *« modularité »* est le plus utilisé. Mais La plupart de ces méthodes partitionnent les nœuds en classes disjointes.

Récemment, des études ont mis en évidence les limites de ces techniques. En effet, elles ont montré qu'un grand nombre de réseaux "réels" contenaient des nœuds connus pour appartenir à plusieurs groupes simultanément. Dans ce contexte une nouvelle voie de recherche c'est imposée, il s'agit de la détection de communautés chevauchantes (recouvrantes) en anglais (Overlapping Community Structure in Networks). Dans ce cas les nœuds peuvent appartenir à plus d'une communauté contrairement aux communautés disjointes ou un nœud appartient à une seule communauté.

La détection de communautés chevauchantes est un problème bien plus difficile que la détection de communautés disjointes. la notion de fonction de qualité ne peut pas s'appliquer directement. Il n'existe pour le moment que très peu d'approches qui traitent ce sujet .

Le travail qui m'a été proposé dans le cadre de ce mémoire est la détection de communautés chevauchantes par filtrage des nœuds d'une décomposition de nœuds d'un réseau biparti en groupes chevauchants. Cette décomposition est obtenue à partir d'une partition des arêtes de ce réseau. Pour améliorer la décomposition obtenue on utilise un algorithme de filtrage des nœuds du réseau. Cet algorithme optimise la modularité du réseau. Pour évaluer l'amélioration de la décomposition on va mesurer sa qualité avant et après le filtrage, grâce à une fonction de qualité MQ_{over} introduite par Mancoridis et al pour des décompositions des nœuds en communautés chevauchantes. Cette mesure est destinée aux graphes mono-partis. L'algorithme qui m'a été proposée l'a adapté aux graphes bipartis. Enfin on va visualiser les communautés avant et après filtrage.

Ce mémoire est structuré comme suit :

- Le chapitre 1 : « Détection de communautés dans les réseaux » présente les graphes complexes et leurs applications dans les différents domaines concernés. Nous introduirons ensuite la problématique de détection de communautés et les différentes méthodes utilisées.
- Le chapitre 2 : « Détection de communautés chevauchantes » présente la détection de communautés chevauchantes, et quelques exemples qui nécessitent ce type de détection ainsi que les méthodes utilisées.
- Le chapitre 3 : « Optimisation de la modularité chevauchante pour la détection de communautés » Dans ce chapitre on présentera la notion de modularité et ses caractéristiques ainsi que la modularité pour des communautés chevauchantes et l'optimisation de cette mesure.
- Le chapitre 4 : « Conception et implémentation » décrit les différents étapes pour implémenter l'algorithme filtrage ainsi que les structures de données utilisés.
- Le chapitre 5 : est dédié à « la mise en œuvre et l'expérimentation » du logiciel, on terminera le rapport sur une conclusion.

Chapitre 1

Détection de communautés dans les réseaux

1. Introduction

De nous jours, les réseaux réels sont devenus de plus en plus grands et complexes, leurs structures montrent une grande variation d'après le problème. Cette structure apparait dans différents domaines comme la biologie, la chimie, l'informatique, etc.

En effet, il est difficile de travailler sur des réseaux réels car ceux-ci sont difficiles et coûteux à construire, et de plus on ne peut pas contrôler leurs propriétés. Une solution à ce problème a été introduite, elle consiste à partitionner le réseau complexe en sous réseaux (en communautés) c'est ce qu'on appel « La détection de communautés », c'est une problématique assez ancienne, qui tire ses origines du problème de partitionnement du graphe qui est un problème classique en théorie des graphes .

2. Notion de réseau complexe [1]

Les réseaux complexes sont définis comme des réseaux dont la structure est irrégulière, complexe et évolutive dans le temps.

Les grands réseaux permettent de décrire des systèmes complexes issus de différents domaines. En effet le travail sur les réseaux complexes nécessite d'utiliser des méthodes statistiques et mathématique dans le cadre de la théorie du graphe. Afin d'étudier et de comprendre la structure des relations entre leurs entités on les modélise

généralement par des graphes, c'est-à-dire par un ensemble de sommets (les entités du réseau) et d'arêtes (les relations entre ces entités).

3. Les graphes complexes

3.1. Définition

Un graphe $G = (S,A)$ permet de modéliser des interactions entre des objets (représentés par un ensemble S de n sommets) reliés par des liens (représentés par un ensemble $A \subset S \times S$ de m paires de sommets nommées arêtes). Ainsi des grands graphes de terrain permettent de modéliser de nombreux phénomènes réels (le terme grand fait référence au nombre important de sommets de ces graphes). Ces graphes peuvent être, selon le contexte, orientés ou non-orientés, pondérés ou non-pondérés.

3.2. Définition sur les graphes [2]

➢ **Définition 1 (Graphe) :**

Soit S un ensemble de $n_S \in N^*$ éléments et A un ensemble de $n_A \in N$ couples d'éléments de S. On appelle graphe G le couple (S,A). Les éléments de S sont appelés sommets du graphe et ceux de A, les arcs ou arêtes du graphe, suivant qu'ils sont orientés ou non.

➢ **Définition 2 (Graphe valué ou pondéré) :**

Soit un graphe $G = (S,A)$. On dit que le graphe est valué (ou pondéré) si à chaque élément a de A est associé une valeur entière $poids(a) \in N^*$.La valeur $poids(a)$ est appelée le poids de a. Par extension, on considère qu'un couple de sommets $(s,s') \in S^2$ tel que $(s,s') \notin A$ possède un poids nul : $poids(s,s') = 0$.

Le poids d'un sous-ensemble X d'éléments de A, est la somme des poids des éléments de X : $poids(X) = \sum_{a \in X} poids(a)$.

➢ **Définition 3 (Adjacence) :**

Soit un graphe $G = (S,A)$ et une arête $a = (s,s') \in$ A. On dit que les sommets s et s' sont les sommets adjacents à l'arête a. De même ,on dit que a est l'arête adjacente aux sommets s et s'.

➢ **Définition 4 (degré d'un sommet) :**

Dans le cas des graphes valués, on redéfinit le degré d'un sommet s ϵ S comme étant la somme des poids des arêtes adjacentes à ce sommet. Ainsi: $deg(s) = \sum_{(s,s') \in A} poids(s,s')$. Pour un graphe quelconque, on peut définir une unique fonction poids : A → N qui à toute arête de ce graphe associe son poids. Cette fonction poids peut être représentée sous la forme d'une matrice. On définit la matrice d'adjacence d'un graphe G comme étant la matrice associant à chaque arête de G son poids.

➢ **Définition 5 (Graphe simple)** :

Un graphe est dit simple s'il n'a ni boucle ni arête multiple.

➢ **Définition 6 (Matrices d'adjacence) :** Soit un graphe simple G = (S,A).

La matrice M_{Adj} de dimensions $n_v \times n_v$, telle que $\forall (i, j) \in \{1, \ldots, n_S\}^2$:

$$(M_{Adj})_{ij} = \begin{cases} 0 & \text{si } i = j \\ p(i,j) & \text{si non} \end{cases}$$

➢ **Définition 7 (Densité) :**

La densité d'un graphe G=(S ,A) est le rapport $|A| / |S|^2$.

- un graphe peut dense contient peu d'arêtes (arcs) .
- un graphe dense contient beaucoup d'arêtes (arcs).

➢ **Définition 8 (graphes bipartis) :**

Un graphe biparti est un graphe qui contient deux familles (types) de sommets disjointes, telles qu'il n'existe aucun lien reliant deux sommets de la même famille.

Autrement dit : G=(S,A) est biparti si et seulement s'il existe une partition S1∪S2 de S tel que ∀(u,v)∈A, u∈S1⇔v∈S2.

➢ **Définition 9 (arête incidente à un sommet) :**

Une arête est dite incidente à un sommet v_i si v_i est une de ses extrémités.

3.3. Quelques domaines concernés [3]

On va présenter quelques exemples dans lesquels des graphes sont utilisés comme outil de modélisation de phénomènes complexes. Ces exemples vont illustrer la diversité des domaines d'applications possibles. Pour chaque cas, nous identifierons les

acteurs du phénomène, modélisés par les sommets du graphe, et les interactions entre eux, modélisées par des liens ou arêtes entre les sommets.

- **Les réseaux sociaux** :

  Ils constituent un champ d'application ancien et important dans lequel les acteurs sont des individus ou entités sociales (associations, entreprises, pays, etc). Les liens entre eux peuvent être de différentes natures. Nous pouvons ainsi observer plusieurs types de réseaux :

  - *Les réseaux de connaissance* : deux individus sont reliés s'ils se connaissent.
  - *Les réseaux de contact physique* : deux individus sont reliés s'ils ont été physiquement en contact.
  - *Les réseaux de collaboration* : deux individus sont reliés s'ils ont travaillé ensemble.
  - *Les réseaux d'appels téléphoniques* : deux individus ou numéros de téléphones sont reliés s'il y a eu un appel entre eux.
  - *Les réseaux d'échanges* :deux entités sont reliées si elles ont échangé un fichier ou un courrier électronique par exemple, etc.

- **Les réseaux biologiques** : Il en existe plusieurs types parmi lesquels nous pouvons citer :

  - *Les réseaux métaboliques* : les sommets sont des gènes ou des protéines qui sont liés par leurs interactions chimiques.
  - *Les réseaux de neurones* : chaque neurone est connecté à plusieurs autres neurones.
  - *Les réseaux trophiques* : les espèces d'un écosystème sont reliées pour représenter les chaînes alimentaire.

- **Les réseaux d'infrastructure** : Ils représentent des connections matérielles entre objets distribués dans un espace géographique. Nous pouvons citer :

  - *Les réseaux de transport* :routes entre villes ou liaisons aériennes entre aéroports.
  - *Les réseaux de distribution électrique* : câbles entre les lieux de production et de consommation .
  - *Le réseau physique de l'Internet* : câbles entre ordinateurs.

- **Les réseaux d'information** : Ils représentent des liens abstraits de référencement entre des supports d'information. Parmi eux :
  - *Les réseaux de citation d'articles* ou *les graphes du Web* :les sommets sont des pages Web liées par des liens hypertextes.

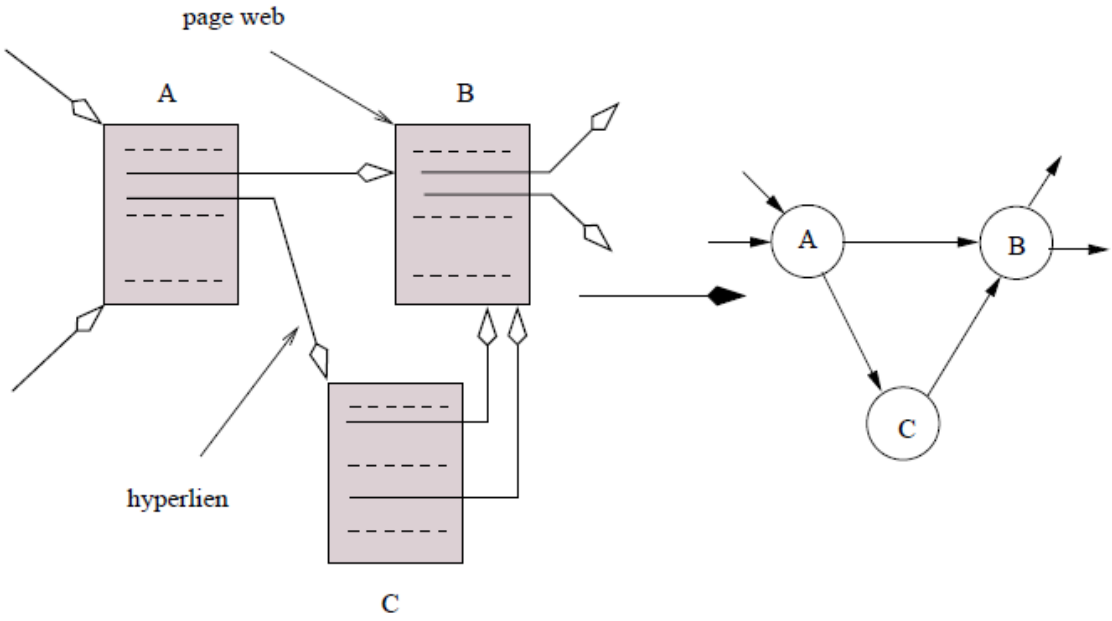


Figure1.1 : Figure d'un graphe qui modélise la structure hypertextuelle du Web.

- **Les réseaux linguistiques :** Ils relient les mots d'un langage donné et regroupent entre autres :
  - *Les réseaux des synonymie* : deux mots sont reliés s'ils sont synonymes.
  - *Les réseaux de cooccurrences* :deux mots sont reliés s'ils apparaissent dans une même phrase d'un ouvrage.
  - *Les réseaux de dictionnaires* : deux mots sont liés si l'un est utilisé dans la définition de l'autre.

3.4. problème lie aux graphes complexes

Avec la croissance du graphe et l'augmentation de ses entités, ceux-ci les rendent de plus en plus complexes et difficile à gérer. De ce fait il est nécessaire de trouver un moyen qui permet de mieux comprendre la structure de ces graphes et qui peut être utilisé comme brique de base dans l'élaboration d'autres traitements plus complexes

(parallélisassions, visualisation ou compression par exemple). Pour cela les chercheurs ont pensé à découper les réseaux en sous ensembles significatifs (les communautés).

4. La détection de communautés

L'étude de structures de communautés dans les réseaux devient de plus en plus une question importante. La connaissance des modules de base (communautés) des réseaux nous aide à bien comprendre leurs fonctionnements et comportements, et à appréhender les performances de ces systèmes.

4.1. Qu'est ce qu'une communauté? [4]

Une communauté dans un graphe (réseau) est définie comme un ensemble de nœuds qui sont fortement liés entre eux, mais faiblement liés avec le reste du graphe. Les membres de la même communauté partagent les mêmes centres d'intérêt.

4.2. Définition de la détection de communautés [5]

La détection de communautés consiste à trouver les communautés dans un réseau donné, sans connaître à priori ni la taille ni le nombre des communautés.
la découpe en communautés joue un rôle clef, non seulement en informatique, mais aussi par ses applications dans de nombreux domaines scientifiques. c'est une problématique assez ancienne, qui tire ses origines du problème de partitionnement du graphe qui est un problème classique en théorie des graphes .

Etant donné un graphe non-orienté **G = (S, A)**, les sommets et les arrêtes peuvent être pondérés, où $|s|$ est le poids du sommet s et $|a|$ est le poids de l'arête a. Le problème du partitionnement de graphe consiste à diviser G en **k** partitionnement disjointes. Au point de vue de mathématique, on peut partitionner les sommets ou bien les arrêtes. Par contre, dans la plupart des applications, on ne s'intéresse qu'au partitionnement des sommets de graphe. Soient un graphe **G = (S, A)** et un ensemble de k sous-ensembles de S, noté Pk = {S1, S2,..., Sk}. On dit que Pk est une partition de G si :

- Aucun sous-ensemble de S qui est élément de Pk n'est vide :
  $\forall i \in \{1, \dots, k\} \quad , Si \neq \emptyset$
- Les sous-ensembles de S qui sont élément de Pk sont disjoins deux à deux
  $\forall i, \mathrm{j} \in \{1, \dots, k\} \quad , Si \neq j, \quad Si \cap Sj = \emptyset$

- L'union de tous les éléments de Pk est S :

$$\bigcup_{i=1}^{k} S_i = S$$

Les éléments *Si* de *Pk* sont appelés les parties de la partition

Dans le cas k = 2, on a le problème de bissection.

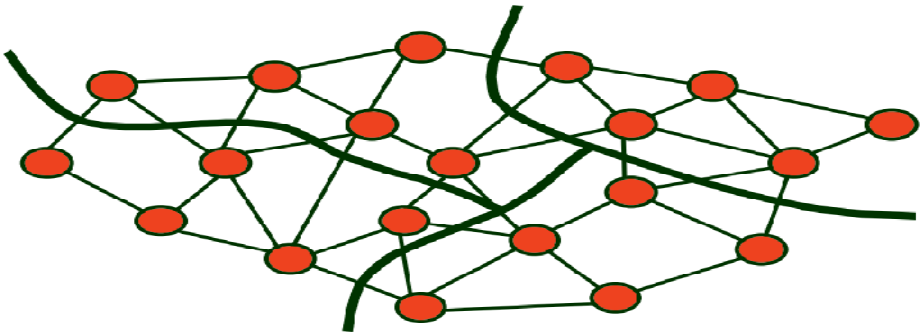


Figure1.2 : Partitionnement de Graphe avec k=4

Comme le montre cette définition on n'a pas a priori de définition précise de la notion de communauté : tout repose sur **la fonction de qualité Q** [3]qui joue donc un rôle clé. La fonction de qualité donne un score à toute partition qui mesure à quel point les communautés de cette partition vérifient des critères caractéristiques des communautés. En particulier, ces fonctions des qualités Q(P) tiennent généralement compte des densités de liens à l'intérieur et entre les communautés. Il existe plusieurs fonctions de qualité pour la détection de communautés, la plus communément utilisée étant la modularité. Les autres fonctions ne sont pas présentées dans le cadre de ce mémoire.

Toutefois, le lecteur intéressé est invité à consulter la référence suivante [3]

4.3. Modularité

La modularité est une mesure utilisée pour évaluer la qualité d'un partitionnement d'un graphe (ou réseau), en communautés. Elle a été introduite par M. E. J. Newman [6]. Le principe est qu'un bon partitionnement d'un graphe implique un nombre d'arêtes intra-communautaires important et un nombre d'arêtes inter-communautaires faible.

4.4. La complexité

D'une manière générale, pour résoudre un problème, on est appelé à trouver l'algorithme le plus efficace. Cette notion d'efficacité induit, selon des critères de temps d'exécution, de nombre d'itérations (ou d'opérations arithmétiques) ou encore d'espace mémoire nécessaire. La complexité permet de mesurer les performances d'un algorithme et de le comparer avec d'autres algorithmes réalisant les même fonctionnalités. Rappels d'algorithmique pour les problèmes de décision [8][9]:

Classe P: problème soluble par un algorithme polynomial déterministe (machine de turing) dont le temps d'exécution est de complexité polynomiale (P signifiant alors *Polynomial time)*.

Classe NP : problème soluble par un algorithme polynomial non-déterministe (machine de Turing non déterministe) dont le temps d'exécution est de complexité polynomiale *(NP* signifiant *Nondeterministic Polynomial time).*

Classe NP-complets : sont les problèmes les plus difficiles de NP. c'est le cas de nombreux problèmes considérés comme intraitables en pratique (pour lesquels les algorithmes déterministes connus sont exponentiels).

Le problème de partition d'un graphe est NP-complet, et il n'existe pas (plus précisément, on ne connaît pas) d'algorithme polynomial déterministe pour le résoudre dans sa forme générale[7].

Conclusion : Il existe plusieurs méthodes de résolution, (voir la section 5. Méthodes de détection de communautés).

4.5. Applications

La découpe en communautés trouve ses applications dans de nombreux domaines scientifiques [5] [10].

- **Modèles épidémiologiques**

La transmission des maladies nosocomiales au sein des CHU est un enjeu majeur de santé publique. De manière évidente: la transmission de ces maladies dépend de la structuration en sous communautés. Les algorithmes de clustering permettent donc de mieux évaluer et de mieux cibler la prévention au sein des services hospitaliers .

- **La protéomique**

Le protéome correspond à l'ensemble des protéines d'une cellule. Les protéines sont des molécules interagissant les unes avec les autres. Les protéines dans la cellule sont regroupées sous forme de complexes fonctionnels. La compréhension de ces complexes a un rôle clef dans les maladies neurodégénératives et dans les processus cancéreux.
Le protéome humain montre qu'une découpe en communautés est nécessaire à une compréhension des graphes d'interactions protéine-protéine.

- **Réseau de téléphonie mobile**

Un graphe des appels en Belgique a été réalisé . La découpe en communautés permet de comprendre, de visualiser les groupes sociaux et de leur proposer des forfaits adaptés à leurs besoins.

- **La conception de circuits intégrés électroniques**

Le problème de conception de circuits intégrés électroniques est de diviser les composants en sous-circuits tels que le nombre de connexion entre ceux-ci soit minimal. C'est une tâche de plus en plus complexe car les composants sont de plus en plus petits et les nombres de composants augment rapidement.
Un graphe peut être utilisé pour représenter un circuit électronique et ce problème devient le problème de partitionnement de graphe. Les sommets représentent les composants et les arrêtes représentent les connexions des composants.

- **La répartition de charge pour les machines parallèles**

La répartition de charge pour les machines parallèles a pour but de répartir les charges de calcul entre les processeurs et de réduire la durée des communications entre les processeurs. Un graphe non-orienté permet de modéliser la répartition des charges pour les machines parallèles. Les sommets du graphe représentent les blocs de calcul et les arrêtes sont correspondants aux communications nécessaires à la programmation parallèles.

- **La segmentation d'images**

La segmentation d'images a pour but d'isoler les différents objets sur une image. Une image est représentée par une matrice de pixels. Il est possible de créer un graphe de l'image, les sommets représentent les pixels et les arrêtes correspondent à la différence d'intensité lumineuse entre les pixels.

- **La classification**

Les outils de classification permettent de trier un ensemble des objets afin de regrouper des objets de même type. Il est donc possible de créer un graphe dont chaque objet est un sommet et chaque arrête exprime la similarité entre deux objets.

5. Méthodes de détection de communautés

Les méthodes de détection de communautés ont fait l'objet de nombreux travaux, par conséquent on peut les classer comme suit :

5.1. Partitionnement contraint et non contraint [3]

1. **Partitionnement contraint :**

Le problème du partitionnement contraint consiste à trouver une partition dont :

- les parties sont de tailles similaires ;
- le coût de coupe est minimisé.

Exemple : Problème de la répartition de charge dans les machines parallèles.

2. **Partitionnement non contraint :**

Trouver une partition qui minimise une fonction objective en tenant compte des différences de tailles entre les parties .

5.2. Les méthodes agglomératives et les méthodes divisives [11]

La plupart des méthodes de détection de communautés se basent sur l'intuition qu'une structure communautaire est par nature hiérarchique, c'est-à-dire qu'une communauté est elle-même composée de sous-communautés qui sont à leur tour composées de sous-sous-communautés et ainsi de suite.

1. **Les méthodes agglomératives** : Ces méthodes tentent de fusionner de manière récursive des petites communautés en de plus grandes en effectuant les choix sur la base d'une mesure de proximité entre les communautés, avec comme point de départ des communautés atomiques ne contenant qu'un sommet.

2. **Les méthodes divisives** : Celles-ci au contraire tentent d'identifier les liens inter-communautaires et de les supprimer pour isoler petit à petit les communautés.

5.3. Les méthodes heuristiques [12] : il en existe deux classes :

1. **Les heuristiques globales :** Elles reposent sur l'exploitation de la globalité de l'information disponible sur le graphe .
2. **Les heuristiques locales :** Dans ce cas, on résout le problème de la *p*-partition sans considérer le graphe dans sa totalité. On étudie les **voisinages** autour d'un sommet ou d'un ensemble de sommets.

Dans ce mémoire nous avons choisi de présenter les méthodes en rapport avec notre travail en les répartissant entre méthodes classiques et méthodes méta-heuristiques.

5.4. Méthodes classiques

5.4.1. Les méthodes d'expansion de région [3]

Les méthodes d'expansion de région (graph growing en anglais), sont des méthodes déterministes. Elles sont très simples à mettre en œuvre et efficaces sur des problèmes de faibles tailles (plusieurs dizaines de sommets). Leur inconvénient est qu'elles sont moins performantes sur des problèmes de plus grandes tailles**.**
Les qualités de ces méthodes font qu'elles sont utilisées de manière sous-jacente par les méthodes multi-niveaux, pour créer des partitions à partir de graphes de tailles réduites. (Le Greedy Graph Growing Algorithm est l'une des méthode d'expansion de région).

- **Greedy Graph Growing Algorithm *(GGGP)***

Le Greedy Graph Growing Algorithm (GGGP) a été introduit dans [6]. Algorithme GGGP est pour résoudre le problème de partitionnement de graphe avec k = 2 (bissection). Il consiste à créer de manière itérative un ensemble E. Cet ensemble est initialisé par un sommet au hasard. A chaque itération, un des sommets adjacents aux sommets de l'ensemble E est ajouté à E. Le sommet sélectionné est le sommet qui peut diminuer le plus le coût de coupe entre E et le reste de G (le coup de coupe est une

fonction objective). Autrement dit, c'est le sommet qui a le gain de la fonction objective le plus grande. Le processus s'arrête si le poids de l'ensemble E est égal la moitié du poids total des sommets de graphe. Cette méthode est très simple et la qualité de la partition obtenue dépend principalement du sommet de l'initialisation. Pour obtenir de meilleur résultat, on peut lancer le programme avec des sommets de l'initialisation différents.

Algorithme d'expansion de région GGGP.

Procédure GGGP (G=(S, A))

Prendre au hasard un sommet $s_0 \in S$.
E→ {s_0}
F→ {s ∈ R tel que (s, s_0) ∈ A} // F est l'ensemble des sommets adjacents à E
// R est l'ensemble des sommets restants
Calculer les gains de F ;
Tant que poids(E) <½ poids(S) ***faire***
Prendre le sommet $s_i \in F$ de gain maximal ;
Déplacer s_i de F vers E ;
Pour toute arête (s, s_i) dans A ***faire***
si s ∈ F ***alors***
Mettre à jours le gain de s
sinon
si s ∉ E ***alors***
Ajouter s à F
Calculer le gain de s ;
fin si
fin si
fin pour
fin tant que
retourner E
fin Procédure

5.4.2. Méthode spectrale [5]

L'utilisation de la méthode spectrale pour résoudre les problèmes de partitionnement de graphe est très ancienne. Les premiers articles sont proposés depuis les années 70s par W.Donath et A.Hoffman. Cette méthode est basée sur le théorème spectral de l'algèbre linéaire. Ce théorème permet d'affirmer la diagonalisation des

matrices réelles. Il justifie également la décomposition des matrices symétriques réelles en valeurs propres dans une base ortho normale de vecteurs propres. Le problème de partitionnement de graphe peut être ramené à la résolution d'un système numérique $Mx = \lambda x$. Résoudre ce système numérique consiste à trouver une base orthogonale de vecteurs propres de la matrice M.

Avantages de la méthode spectrale

- permet de trouver des solutions au problème du partitionnement contraint et non contraint ;
- permet de trouver une borne inférieure au coût de coupe de partitionnement d'un graphe non pondéré.

Inconvénient

Mieux adapté à la bissection qu'au k-partitionnement.

5.4.3. Algorithme de Kernighan-Lin [5]

L'algorithme de Kernighan-Lin n'est pas exactement une méthode de partitionnement mais plutôt une méthode d'affiner une bissection d'un graphe précédemment obtenu. L'idée principale de cet algorithme est de trouver deux sous-ensembles de sommets de même taille, chacun dans une partie de la bissection, tels que leur échange diminue le coût de coupe de la bissection. A partir d'une bissection existante, l'algorithme échange successivement deux sous-ensembles de la bissection jusqu'à on ne peut pas diminuer le coût de coupe. La dernière bissection obtenue est la bissection de coût de coup minimal trouvé par l'algorithme.

L'algorithme de Kernighan-Lin est d'autant plus rapide et performant si sa partition initiale est de bonne qualité, dans ce cas le nombre d'itérations de l'algorithme sera petit car il sera rapidement impossible de trouver des sous-ensembles à échange. De plus, l'optimisation trouvée est locale. Ainsi, pour avoir un coût de coupe minimal, il est préférable de partir d'une bissection de coût de coupe faible. On peut utiliser la méthode de GGGP pour créer la bissection initiale.

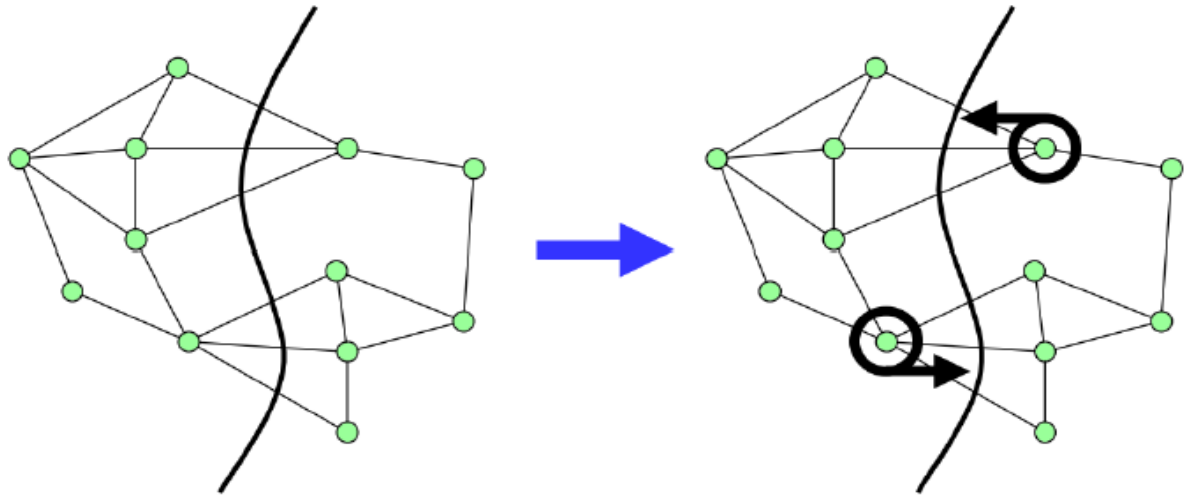


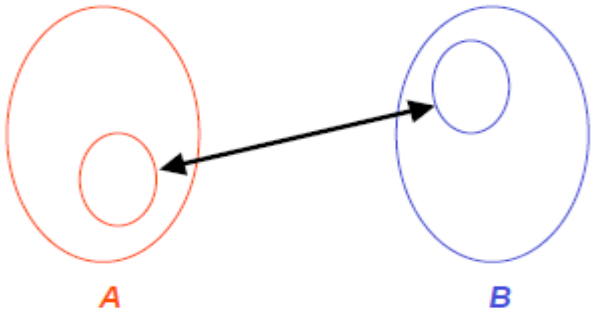
Figure1.3 : L'idée principale de l'algorithme Kernighan-Lin

Idées à la base [7] :

– On démarre avec une **bissection arbitraire** de *V* en deux sous-ensembles *A* et *B* tel que | *A* | = | *B* | (à un près).

– On essaie d'**améliorer** la bissection initiale en **échangeant** un **sous-ensemble de *A*** avec un **sous-ensemble de *B***.

Ces sous-ensembles sont déterminés à partir d'une **fonction de coût**.



5.4.4. La méthode multi-niveaux [5][3]:

La méthode multi-niveaux a été parallèlement utilisée pour la première fois par Stephen Barnard et Horst Simon et par Thang Bui et Curt Jones en 1993. Elle est actuellement la méthode la plus utilisée pour résoudre des problèmes de partitionnement de graphe contraint.

La méthode multi-niveaux cherche à répondre à une question simple : comment créer rapidement une partition d'un graphe G de grande taille, sachant qu'il est très coûteux de s'occuper de chaque sommet un par un ?

La réponse naturelle à cette question est de regrouper les sommets ensemble pour s'occuper de groupes de sommets plutôt que de sommets indépendants. C'est cette idée qui est à l'origine de la méthode multi-niveaux.

La méthode multi-niveaux se décompose en trois phases bien distinctes qui agissent chacune sur le graphe G :

1. **Contraction** : la phase de contraction est de nature itérative. À chaque itération, les sommets du graphe résultant de l'itération précédente sont regroupés pour former un graphe similaire, mais dont le nombre de sommets est plus petit que le précédent.

2. **Partitionnement** : le but de cette étape est de créer une partition P du graphe G_n. Pour cela, le graphe G_n résultant de l'étape de contraction peut être partitionné en utilisant une heuristique de partitionnement comme par exemple une méthode d'expansion de région.

3. **Affinage** : l'étape d'expansion, durant laquelle la partition initiale est projetée et raffinée sur les graphes de plus en plus gros jusqu'au graphe initial.
affinée en utilisant un algorithme local d'affinage de type Kernighan-Lin.

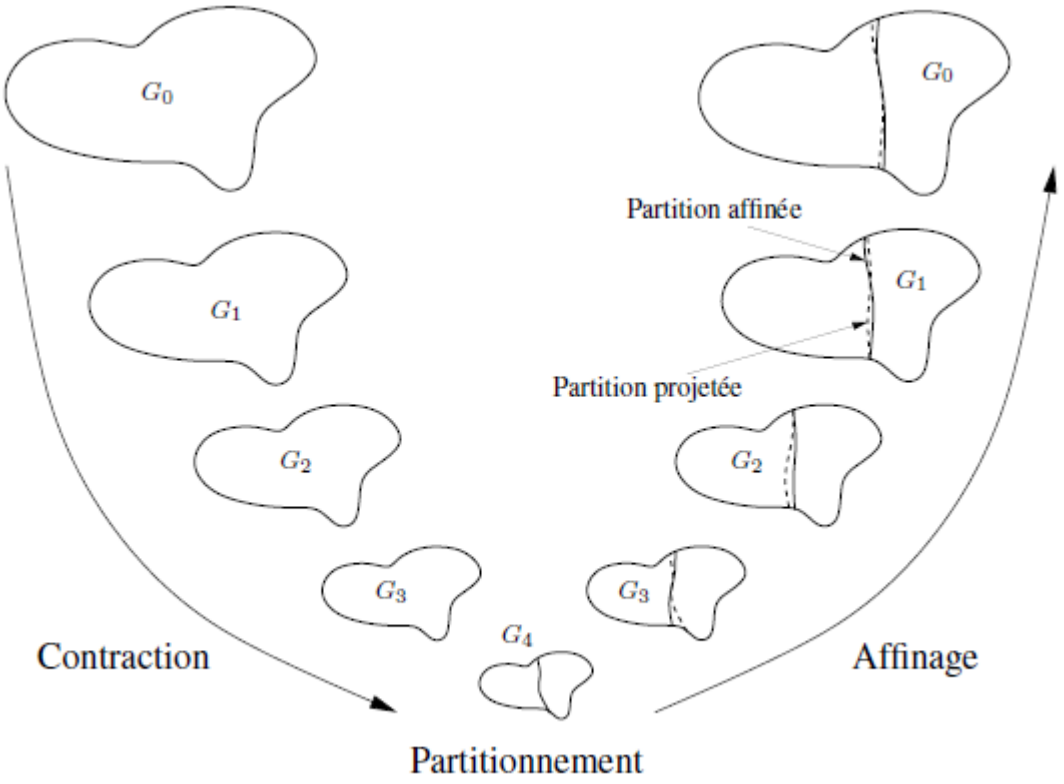


Figure1.4 : La méthode multi-niveaux

5.5. Les méthodes Métaheuristiques

Les métaheuristiques sont apparues dans les années 1980 pour résoudre des problèmes d'optimisation difficile. Ces méthodes ont en commun certaines caractéristiques [13] :

- Leur nature stochastique leur permet d'explorer plus facilement un espace des solutions de très grande taille qui étaient impossible de les traiter auparavant .
- Elles sont inspirées par des analogies avec la nature.
- Elles sont en général non-déterministes et elles sont coûteuses en temps de calcul.
- Le principe de base d'une métaheuristique est de parcourir l'espace des solutions afin de déterminer des solutions (presque) optimales ,et elles peuvent contenir des mécanismes qui permettent d'éviter d'être bloqué dans des régions de l'espace de recherche (*les optimums locaux*).
- L'un des atouts majeurs des métaheuristiques est leur facilité d'adaptation à de nouvelles fonctions objectifs.

5.5.1. Algorithme du recuit simulé [3]

La métaheuristique du recuit simulé s'inspire de la technique expérimentale du recuit utilisée en métallurgie. Celle-ci permet d'obtenir un état stable, *ou d'énergie minimale*, du métal. Cet état est obtenu quand le matériau a trouvé une structure cristalline. Alors que la technique bien connue de la trempe fige le matériau dans un état méta stable, la technique du recuit permet d'éviter au matériau d'être dans un état méta stable caractérisant *un minimum local* d'énergie. Il consiste à porter le matériau à haute température, puis à abaisser lentement celle-ci.

En informatique, la méthode du recuit simulé transpose cette technique à la résolution d'un ***problème d'optimisation***. Ainsi, *la température*, paramètre essentiel du *recuit physique*, est directement utilisée dans la méthode du *recuit simulé* comme *paramètre de contrôle*. De même, par analogie avec le processus physique, *l'énergie du système* devient la *fonction de coût à minimiser*.

L'algorithme du recuit simulé est décrit dans de nombreux ouvrages. Son principe est simple : partant d'un état initial quelconque, un nouvel état est créé à partir de l'état

précédent par une modification élémentaire de cet état. Il est accepté si son énergie est plus faible, sinon il est accepté avec une certaine probabilité.

5.5.2. La recherche tabou [14]

La méthode tabou a été développée par Glover . Pour certains chercheurs cette méthode apparaît plus satisfaisante sur le plan scientifique que le recuit simulé, car la partie « aléatoire » de la méthode a disparu.

La méthode tabou partage avec l'algorithme du recuit simulé l'idée de guider la recherche local pour éviter les optimums locaux. Contrairement au recuit simulé qui génère de manière aléatoire une seule solution voisine **s'** dans le voisinage **N(s)** à chaque itération, la méthode tabou examine un échantillonnage de solutions **N(s)** et retient la meilleure **s'** même si **s'** est plus mauvaise que **s** . La recherche tabou ne s'arrête donc pas au premier optimum trouvé. Cependant cette stratégie peut entraîner des cycles, par exemple un cycle de longueur 2 : **s** ⟶ **s'** ⟶ **s** ⟶ **s'**....

pour empêcher ce type de cycle, on mémorise les **k** dernières configurations visitées dans une mémoire à court terme et on interdit tout mouvement qui conduit à une de ces configurations. Cette mémoire est appelée la liste tabou (qui a donné le nom de la méthode). La valeur de k dépend du problème à résoudre et peut éventuellement évoluer au cours de la recherche.

5.5.3. Algorithme de colonies de fourmis [15]

La méthode de colonie de fourmis s'inspire du comportement des colonies de fourmis réelles. Le principe de cette méthode est le suivant :

Malgré la vision très limitée de chaque fourmi, une colonie de fourmis parvient a minimiser la longueur du chemin conduisant à une source de nourriture, grâce aux traces chimiques (phéromones) laissées par chacune des fourmis. Un principe analogue a été utilisé pour traiter le problème du voyageur de commerce ainsi que d'autres problèmes d'optimisation. La méthode consiste à réitérer un algorithme de construction (assimilé à l'action d'une fourmi) dans lequel chacun des choix est déterminé en tenant compte des traces laissées par les fourmis précédentes.

Une fourmi qui a emprunté une arête incite les fourmis suivantes à emprunter cette même arête à leur tour. Les fourmis peuvent collectivement trouver le plus court

chemin entre deux points. Cette méthode a été adaptée pour le problème de partitionnement de graphe ces dernières années.

Il faut noter que la méthode des colonies de fourmis est souvent hybridée avec la recherche locale.

5.5.4. Algorithme évolutionnaire [14]

Les algorithmes évolutionnaires sont des techniques d'optimisation itérative.

Un algorithme évolutionnaire simule un processus d'évolution sur une population d'individus, dans le but de faire évoluer vers les optimums globaux du problème d'optimisation considéré. Il réunit trois composants :

1- Une population constituée de plusieurs individus représentant des solutions potentielles au problème posé (en optimisation combinatoire classique : configuration du problème).

2- Une fonction d'adaptation (fitness) qui évalue la performance d'un individu par rapport au milieu (en optimisation combinatoire : fonction du coût).

3- Un mécanisme d'évolution de la population composé de plusieurs opérateurs de modification et de sélection permettant (grâce à ces opérateurs prédéfinis), d'éliminer certains individus et d'en créer de nouveaux.

6. Conclusion

Dans les grands réseaux, la détection de communautés, est un problème que l'on retrouve dans plusieurs disciplines . Ces communautés jouent un rôle important dans l'organisation ou la structuration des réseaux, permettent de manipuler des graphes particulièrement grands.

En effet, il s'agit de déterminer des classes dans un graphe: suivant l'usage que l'on veut faire de ces communautés, les classes peuvent (doivent) être disjointes ou bien chevauchantes.

Dans le chapitre suivant on va étudier la détection de communautés chevauchantes.

Chapitre 2

Détection de communautés chevauchantes

1. Introduction

Les réseaux sont largement utilisés afin de décrire les interactions entre les entités. Dans ce contexte, de nombreuses méthodes ont été développées afin d'extraire des informations, à partir de la topologie des réseaux. La plupart d'entre elles partitionne les nœuds en des classes disjointes.

Récemment, des études ont mis en évidence les limites de ces techniques.
En effet, elles ont montré qu'un grand nombre de réseaux "réels" contenaient des nœuds connus pour appartenir à plusieurs groupes simultanément.

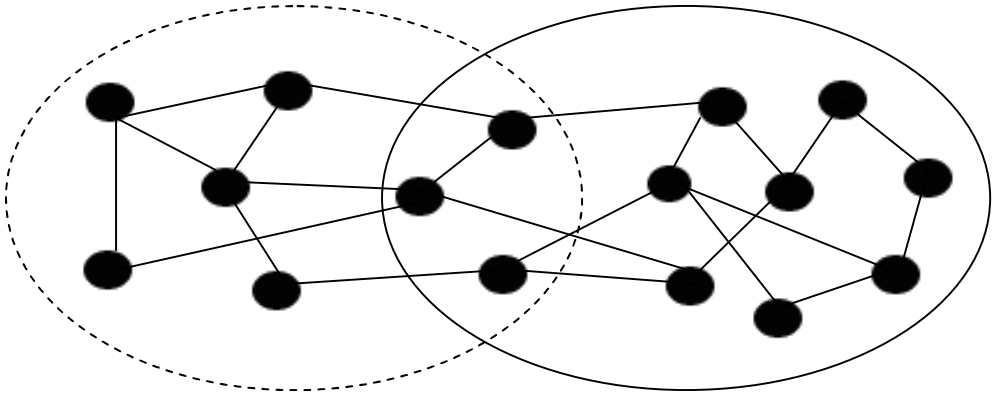
Pour répondre à ce problème, une nouvelle approche a été introduite il s'agit de « la détection de communautés chevauchantes », en anglais (Overlapping Community Structure in Networks).

2. Définition de communauté chevauchante

Les nœuds peuvent appartenir à plus d'une communautés, contrairement aux communautés disjointes ou un nœud appartient à une et une seule communauté.

Définition mathématique[16] :

Deux ensembles X et Y se chevauchent si seulement si : $X \cap Y \neq \emptyset$ et $X \setminus y \neq \emptyset$ et $Y \setminus X \neq \emptyset$. On note X ⓪ Y .
X ,Y sont chevauchants c'est-à- dire non égales et d'intersection non nulle.



—— **communauté1**

- - - - - **communauté2**

Figure2.1: Illustration du concept du chevauchement des communautés.

3. Nécessité d'avoir des communautés chevauchantes

Les méthodes citées dans le premier chapitre ont l'avantage d'être efficaces et de s'appliquer à de grands graphes. Mais elles produisent des partitions en classes disjointes. Ce qui n'est pas toujours désiré ni justifié. En particulier dans les réseaux sociaux, en Biologie...etc.

3.1. Quelques exemples

- **Biologie**

En Biologie, où l'on analyse les réseaux d'interactions protéine-protéine pour, entre autres, prédire leurs fonctions, nombreuses sont celles qui ont plusieurs fonctions. Elles sont qualifiées de multifonctionnelles et permettent de comprendre la complexité de certains phénotypes ou les effets secondaires de certaines drogues.

Si l'on cherche à déterminer des classes fonctionnelles, il est injustifié d'affecter chaque protéine à une seule classe, et dans ce cas il est raisonnable de construire non pas une partition, mais un recouvrement, c'est-à-dire un système de classes chevauchantes.

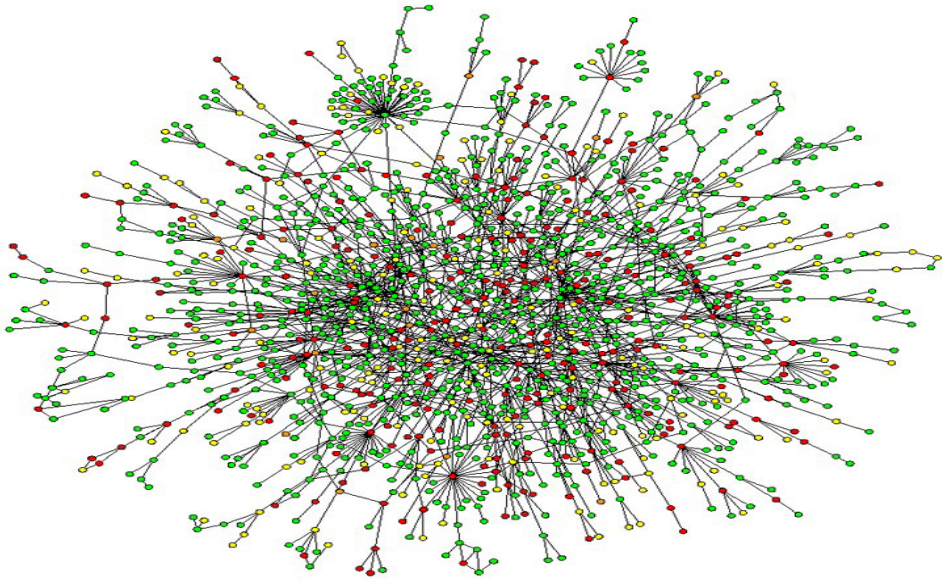


Figure2.2 : graphes d'interactions protéine-protéine.

- **Sociologie**

Un individu peut appartenir à plusieurs groupes (de travail ou de relation), parfois les chercheurs appartiennent à plus d'un groupe de recherche [18].

- **Les systèmes P2P**

Les communautés sont chevauchantes dans le sens où un super-pair peut appartenir à une ou plusieurs communautés [17].

Par exemple, un nœud A appartenant à une communauté de domaine de chimie, il est intéressé par le domaine de Biologie, le nœud A diffuse son intérêt sur le réseau. Après cette déclaration ce nœud devient membre d'une communauté de domaine Biologie.

Par la suite le nœud A appartient à deux communautés ,il chevauche entre les deux communautés (le domaine de chimie et de biologie).

D'où la nécessité de construire des partitions en classes chevauchantes, aussi appelées des *systèmes de classes*.

4. Caractéristiques de la détection de communautés chevauchantes

- La détection de communautés chevauchantes est un problème bien plus difficile que la détection de communautés disjointes.
- Il n'existe pour le moment que très peu d'approches qui traitent ce sujet [19][20]
- Il n'existe pas de formalisme qui permette de définir clairement la notion de communautés qui se recouvrent.
- La notion de fonction de qualité ne peut pas s'appliquer directement.
- Les communautés qui se chevauchent correspondent cependant à une réalité des grands graphes de terrain qui n'est absolument pas traitée par la majorité des approches.

5. Les méthodes de détection de communautés chevauchantes

Il existe plusieurs façons de détecter la structure de communautés chevauchantes dans un réseau :

L'algorithme FOCAL [24] proposé par (Pissard 2008) (Fast Overlapping Clustering ALgorithm) qui restitue des communautés recouvrantes. Son approche est intéressante car elle tient compte des caractéristiques structurelles des réseaux sociaux (petits mondes, transitivité) des communautés.

L'algorithme SCAN (Xu et al 2007) permet aussi de détecter des communautés recouvrantes. Cet algorithme forme des communautés en se basant sur l'idée de base que la structure communautaire d'un nœud est définie par ses voisins .

Dans ce mémoire on va étudier : Un Algorithmes d'optimisation de la modularité pour la détection de communautés chevauchantes dans un graphe, il s'agit de la **méthode fusion**, on va voir aussi d'autres algorithmes, ainsi qu'une mise en œuvre d'une approche de détection de communautés chevauchantes sur un graphe biparti en utilisant un algorithme de détection de communautés disjointes.

➢ **Quelques algorithmes de détection de communautés chevauchantes**

Dans cette partie on va étudie certains algorithmes connus pour la détection de communauté chevauchantes.

5.1. Classification par densité

Cette méthode avait été élaborée dans le cadre de la thèse de T. Colombo [25] et reprise dans un article de L. Denoeud et al [26]. Dans cette méthode on définit une ***fonction de densité*** à chaque sommet du graphe. Celle-ci correspond aux taux d'arêtes ou de triangles dans un voisinage plus ou moins grand autour de chaque sommet.

La stratégie générale consiste à construire les classes autour des ***maxima locaux*** de cette fonction de densité. L'affectation se fait de proche en proche, tant qu'il n'y a pas ambigüité. Les classes se développent en parallèle autour de ces *noyaux* et un sommet adjacent à une seule classe courante lui est attribué. Mais s'il est adjacent à plusieurs classes, on a le choix entre l'affectation à une seule d'entre elles, celle à laquelle il est le plus connecté, ou à toutes, créant ainsi des classes chevauchantes.

Plusieurs fonctions de densité ont été testées et les résultats pratiques, sur des graphes dont les arêtes correspondent à la relation entre certaines types de gènes, sont assez satisfaisants. Mais sur les graphes d'interactions, qui sont très peu denses, ils n'étaient guère encourageants. C'est pourquoi ils ont étendu cette méthode à la construction de classes chevauchantes.

➢ **Définition d'une clique**

Un graphe est dit **clique** s'il est un graphe simple pour lequel il existe **exactement** une arête entre toute paire de sommet.

5.2. Algorithme de fusion

L'algorithme fusion[18] est un algorithme d'optimisation de la modularité pour la détection de communautés chevauchantes dans un graphe. La fusion est le simple remplacement des deux classes par leur union. Partant d'un système de classes chevauchantes et en appliquant uniquement des fusions, on aboutit à des classes qui seront nécessairement chevauchantes, si le système initial l'est.

Deux systèmes initiaux ont été étudiés :

1. Les cliques maximales du graphe
2. Les arêtes du graphe

➢ Les cliques maximales du graphe

Dans la mesure où elles sont énumérables en un temps raisonnable, elles constituent un système de classes chevauchantes dont la modularité est égale à ***Qmax***. Toute opération de fusion fera décroître ***Q***.

➢ Les arêtes du graphe

Avec les arêtes du graphe, on part de la même valeur de modularité ***Qmax*** qu'avec les cliques. Naturellement, le processus de fusion commence par reconstruire certaines cliques, dans la mesure où la fusion des arêtes **(*x, y*) et (*y, z*)** ne coûte rien si **(*x, z*)** est également une arête. Dès que l'algorithme ne trouve plus une paire de classes ***Vi*** et ***Vj*** telle ***que*** $\forall x \in Vi, \ \forall y \in Vj, (x, y) \in E$, la modularité commence à décroître.

L'efficacité de l'algorithme ascendant dépend du nombre de classes initiales, puisque celui-ci détermine le nombre d'itérations. En partant des cliques ou des arêtes, un grand nombre d'itérations est effectué**. Pour réduire le nombre de classes de départ**, les listes d'adjacence ont été utilisées, sans résultats satisfaisants.

Solution : Un système de cliques centrées .

✓ Un système de cliques centrées

Consiste à ajouter les sommets dans l'ordre des degrés relatifs, tant qu'ils réalisent une clique. Une clique, la plus grande possible, est construite en chaque sommet ***x*** du graphe. Cette opération est réalisée par un algorithme polynomial appliqué en chaque sommet qui réalise, en un nombre d'étapes borné par ***n***. La procédure, détaillée dans
l' algorithme qui calcule l'ensemble des cliques centrées [18] .

Remarques :

1. Une simulations est faite Pour vérifier que cette méthode retrouve bien les systèmes de classes chevauchantes .pour plus de détaille consulter [18].
2. Pour optimiser la modularité de cet algorithme des modifications sont ajoutés. c'est ce qu'on va voir dans le chapitre suivant.

5.3. CFinder

Cette méthode est basée sur une recherche de motifs locaux, elle permet un recouvrement des communautés. Elle est élaborées par Palla et autres[21]. (Elle est appelée aussi Percolation de clique).Une communauté est définie comme une **chaîne** de **k-cliques adjacentes**. Une **k-clique** est un sous-ensemble **de k sommets tous adjacents** les uns aux autres, et deux **k-cliques sont adjacentes si elles partagent k - 1 sommets.**

L'avantage immédiat d'une telle approche est la détection de communautés avec recouvrement, un sommet pouvant appartenir à plusieurs k-cliques non forcément adjacentes. Une limite de cet algorithme est qu'il nécessite un paramétrage : la valeur de k (la taille des communautés à considérer).

les chercheurs choisissent une valeur de k = 4, généralement acceptée comme la plus efficace. Les tests qu'ils ont menés semblent montrer que modifier cette valeur influe fortement sur les résultats trouvés mais peu sur la tendance générale de ces résultats comparés aux autres algorithmes.

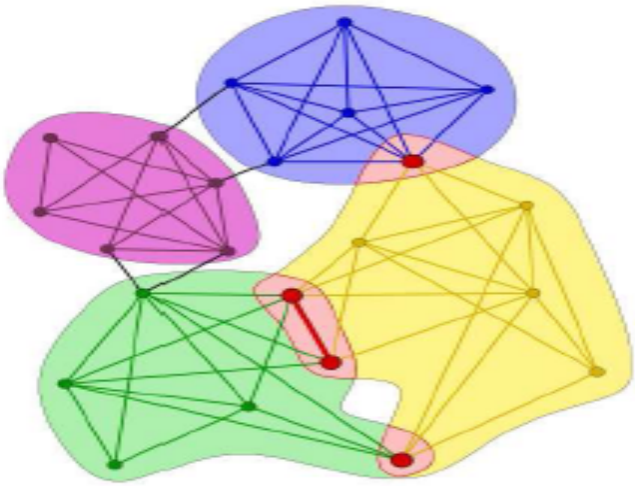


Figure2.3 : Méthode de percolation de clique.

La figure2.2 illustre Les expositions d'exemple des communautés enjambées par 4-cliques adjacent. La communauté jaune recouvre avec la bleue dans un nœud simple, tandis qu'il partage trois nœuds et un lien avec la communauté verte. Ces régions de recouvrement sont soulignées dans le rouge. Sommets de recouvrement sont montrés par les points plus grands ayant la couleur rouge .

5.4. L'algorithme RaRe

L'algorithme de déplacement (RaRe) [22] commence en rangeant tous les nœuds selon un certain critère. Des nœuds fortement rangés sont alors enlevés dans les groupes jusqu'à ce que des petits composants reliés sont formés (appelé les noyaux cluster).Ces noyaux sont alors augmentés en ajoutant chaque nœud enlevé à tout cluster dont la densité est améliorée lorsqu'on ajoute ce dernier.

Tandis que cette approche était réussie en découvrant des clusters, son inconvénient principal était son inefficacité. C'était dû en partie du fait que les rangs et les composants reliés doivent être recalculer à chaque fois que les nœuds sont enlevés. Le temps d'exécution de RaRe est sensiblement amélioré quand les rangs sont calculé seulement une fois.

Amélioration de cet algorithme

5.5. L'algorithme LA

Cet algorithme[22] se concentre sur l'efficacité, Le pseudo-code est donné ci-dessous. Les nœuds sont commandés selon un certain critère. Un nœud est ajouté à tout cluster si l'ajout améliore sa densité. Si le nœud n'est pas ajouté à n'importe quel cluster, il crée un nouveau cluster. Chaque nœud est dans au moins un cluster, donc on peut avoir un nœud dans plusieurs clusters.

Le temps d'exécution peut être lié en termes de nombre de cluster C .

Algorithme LA

```
procédure LA(G = (V,E),W)
 C ← Ø ;
 Order the vertices v1, v2, . . . , v |V |;//nœuds de rang
 for i = 1 to |V | do
   added ← false;
   for all Dj ∈ C do //pour tout cluster
    if W (Dj ∪ vi) > W(Dj) then
      Dj ← Dj ∪ vi ;
      added ← true ;
```

if added = false **then**

C ← C ∪ {{vi}}; *//crée un nouveau cluster*

return C;

La plupart des méthodes sont dédiées pour des réseaux mono-partis. Toutefois, de nombreuses recherches sont consacrées, ces dernières années, afin de trouver des nouvelles approches pour la détections de communautés chevauchantes dans les réseaux bipartis.

5.6. Détection de communautés chevauchantes dans des graphes bipartis [23]

Un graphe biparti est une catégorie spéciale de graphe, où les nœuds peuvent être divisés en deux sous-ensembles disjoints, tels qu'il ne peut y avoir aucun lien qui relie deux nœuds du même sous ensemble. Les réseaux auteur-publication, les réseaux acteur-film, des réseaux produit-consommation, etc. sont des exemples concrets dans la catégorie des réseaux bipartis.

L'approche sur laquelle nous allons se baser pour détecter des communautés chevauchantes dans les réseaux bipartis, se base sur un algorithme appliqué pour détecter des communautés disjointes. Toutefois, dans ce cas l'algorithme se base sur le partitionnement d'arrêts.

Le principe général de cette approche consiste à transformer d'abord le graphe biparti G, en un autre graphe de lien G'. Dans lequel, les sommets désignent les numéros des arêtes de G et les arêtes représentent les liens d'incidence entre les arêtes de G. Puis, en appliquant l'algorithme, pour partitionner le graphe de lien obtenu en sous ensembles. Et enfin, en se basant sur les communautés du graphe G' trouvé, on déduit la structure en communautés chevauchantes des sommets pour le graphe G.

5.6.1. Construction du graphe de lien

Pour construire le graphe G' dont les sommets sont les numéros des arêtes de G et dont les arêtes sont les liens d'incidence entre les arêtes de G, on suit les étapes suivantes :

-construire la matrice A[n, m] du graphe biparti G, dans laquelle, les indices de colonne représentent l'ensemble des nœuds de la première famille et les indices de ligne représentent l'ensemble des nœuds de la deuxième famille ,

D'où : m désigne le nombre total des nœuds de la 1ère famille et n le nombre total des nœuds de la 2ère famille.

$$A[i,j] = \begin{cases} 1 & \text{si } e(i,j) \in E \\ 0 & \text{sinon} \end{cases}$$

Tel que :

e(i, j): Étant l'arête qui relie les sommets i et j de G.

- calculer le nombre d'arêtes du G(V,E) → |E| qui représentent par la suite le nombre de nœuds du graphe G'(V',E') → |V'|, en appliquant cette formule :

$$|E| = |V'| = m_a = \sum_i \sum_j A[i,j]$$

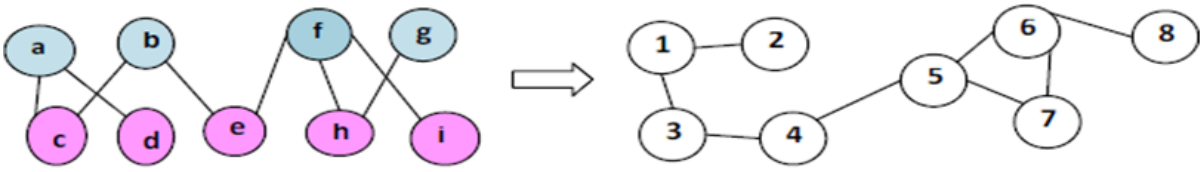
-Numéroter les arêtes du graphe G de k=1 ,… m_a dans la matrice A[n,m] de G :

$$A[i,j] = \begin{cases} k & \text{si } e(i,j) \in E \\ 0 & \text{sinon} \end{cases}$$

- Construire la matrice $A_E[m_a, m_a]$ qui est la matrice d'adjacence du graphe G´ telle que :

$$A_E[i,j] = \begin{cases} 1 & \text{si arete i est incidente à arete j} \\ 0 & \text{sinon} \end{cases}$$

La figure suivante montre un exemple de transformation d'un graphe de nœud au graphe de lien :



Graphe d'origine

Graphe d'incidence des arrêts

Figure2.4: transformation d'un graphe de nœud au graphe de lien.

5.6.2. Partitionnement du graphe de lien

Le partitionnement de E consiste à trouver un ensemble P de N_c clusters tel que $P= \{C_1, \dots C_{Nc}\}$ où les clusters Ci sont disjoints, c'est-à-dire, $(\forall i \neq j,\ C_i \cap C_j = \emptyset)$ et P recouvre E. Autrement dit on a : $\bigcup_i C_i = E$), pour i=1,... N_c.

On va donc appliquer directement l'algorithme génétique, pour partitionner le graphe G' (graphe de lien)

Dans ce cas le problème de chevauchement des arêtes ne se pose pas car une arête représente un lien, et ne peut par conséquent pas appartenir à plus d'une communauté de liens.

La figure3.5. montre un exemple de partitionnement de graphe de lien présenté dans la figure3.4. en deux communautés disjointes.

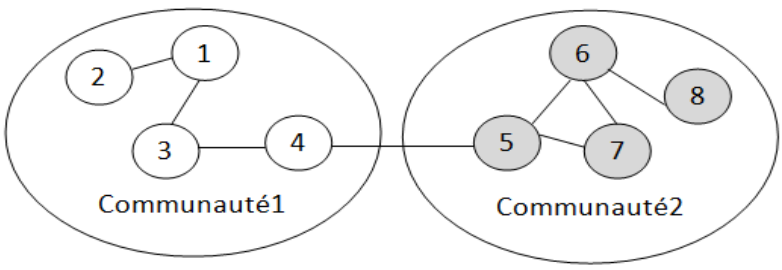


Figure2.5: exemple de partitionnement de graphe de lien en communauté disjointes.

5.6.3. Déduction de communautés chevauchantes

Après avoir partitionné le graphe de lien G' en communauté disjointe, il ne reste qu'à déduire l'ensemble de communautés des nœuds pour le graphe G. La réalisation de ce processus se base sur les communautés trouvées en partitionnant l'ensemble E et sur la matrice adjacence de G après le nommage des liens.

Le nombre de communauté qu'on aura, sera égal au nombre de communautés de G', à la différence que dans ce cas les communautés obtenues sont des communautés chevauchantes.

Les communautés chevauchantes (pour le graphe de nœud), obtenant à partir des communautés disjointes (pour le graphe de lien), sont présenté dans la figure3.6.

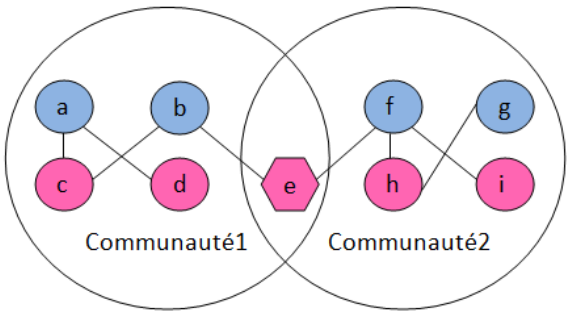


Figure 2.6 : communautés chevauchantes du graphe biparti présente dans la **Figure 2.4**

Remarque : C'est à ce niveau que se situe notre travail. Il consiste à appliquer un algorithme de filtrage à ces communautés chevauchantes pour améliorer le partitionnement.

6. Conclusion

La détection de communautés est un domaine de recherche extrêmement actif et fécond. Une perspective importante est aussi la prise en compte des possibilités de chevauchement des communautés**.** Ce problème est bien plus difficile que la détection de communautés disjointes ,car on ne comprend encore pas bien les communautés qui se chevauchent. outre il n'existe que très peu d'approches qui traitent ce sujet .

Chapitre 3

Optimisation de la modularité pour la détection de communautés chevauchantes

1. Introduction

La décomposition d'un graphe en communautés consiste à partitionner l'ensemble des nœuds du graphe de sorte que les nœuds soient regroupés en zones denses mais avec peu de liens vers l'extérieur.

Pour évaluer la qualité d'une partition, on utilise généralement une fonction donnant un score au partitionnement . Ensuite, il reste à maximiser cette fonction pour trouver la meilleure partition. Malheureusement, la maximisation de cette fonction est un problème NP-complet .Il existe plusieurs fonctions de qualité, la plus utilisée étant la *modularité.*

2. Evaluation de la qualité des communautés

➢ Définition de la décomposition

Considérons un graphe G = (V,E). Le partitionnement de l'ensemble des arêtes E va définir des groupes de nœuds chevauchants qu'on appelle « décomposition ».

Soit une décomposition Cm des nœuds d'un réseau en k groupes $Cm = \{Cm_1, Cm_2, \dots, Cm_k\}$, pour évaluer cette décomposition nous utiliserons la modularité de Mancoridis.

2.1. Définition de la modularité de Newman

La modularité est décrite comme la proportion des arêtes incidentes sur une classe donnée moins la valeur qu'aurait été cette même proportion si les arêtes étaient disposées au hasard entre les nœuds du graphe.

La modularité est une mesure qui a été introduite par NEWMAN en 2004 [27] pour mesurer la qualité d'un découpage en communautés. Elle a été largement utilisée dans les études récentes. Elle représente la métrique de qualité pour l'évaluation du partitionnement d'un réseau en communautés.

- Posons, pour un nœud **v**, $\mathbf{c_v}$ la communauté de **v**.
- Posons **m** le nombre total de lien du graphe **G**, tel que :

$$\mathbf{m} = \frac{1}{2}\sum_{vw} \mathbf{A_{vw}}$$

Où : $\mathbf{A_{vw}}$ vaut 1 s'il existe une arête entre **v** et **w**, 0 sinon.

L'expression de la modularité la plus facilement manipulable est :

$$\mathbf{Q} = \sum_{i} e_{ii} - a_i^2$$

Dont:

$$e_{ii} = \frac{1}{2\mathbf{m}}\sum_{vw} A_{vw}£(C_v,i)£(C_w,i)$$

£($\mathbf{c_w}$,i) : Cette écriture veut dire que le nœud **w** appartient à la communauté **i**. Ou de manière plus intuitive: $\mathbf{e_{ii}}$ est le poids (normalisé) des arêtes dans la communauté **i.**

$$\mathbf{a}_i = \frac{1}{2\mathbf{m}}\sum_{\mathbf{v}} \mathbf{K_v}£(\mathbf{C_v}, \mathbf{i})$$

Où $\mathbf{k_v} = \sum_{\mathbf{w}} \mathbf{A_{vw}}$ ou formulé autrement, $\mathbf{k_v}$ est le degré de **v.**

$\mathbf{a}_i$ est donc le poids total (normalisé) des arêtes connectées à des nœuds de la communauté **i**.

Les algorithmes qui se basent sur cette modularité, ne nécessite aucune connaissance, ni sur la taille, ni sur le nombre des communautés. Il y'a lieu de chercher le « meilleur découpage en communautés » en maximisant la modularité du réseau. Par rapport aux autres méthodes, celle-ci nécessite moins de calcule, ce qui rend les algorithmes utiles dans les réseaux réels de grande taille.

2.2. Définition de la modularité de Mancoridis

La modularité MQ (Modularity Quality), est une mesure introduite par Mancoridis [28] pour mesurer la qualité d'une partition de nœuds du réseau. Cette métrique repose sur la différence entre les ratios de connectivité interne et externe des communautés. Elle est définie comme suit par l'équation 1 :

$$MQ = \frac{1}{k}\sum_{i=1}^{k} S(C_i, C_i) - \frac{1}{k(k-1)}\sum_{i,j=1\ i\neq j}^{k} S(C_i, C_j) \qquad (1)$$

Avec $S(C_i, C_j) = \frac{E(C_i, C_j)}{|C_i|*|C_j|}$ tel que $E(C_i, C_j)$ est le nombre d'arêtes entre les sommets des classes C_i et C_j.

L'équation (1) peut s'écrire sous la forme $MQ = MQ^+ - MQ^-$ avec -1≤ MQ≤ 1.
MQ+ et MQ^- représentent respectivement la cohésion interne et la cohésion externe des classes.

2.3. Caractéristique de la Modularité [2]

- Mesure de qualité d'une partition sur un graphe statique.
- Recherche de groupes denses avec peu de liens à l'extérieur.
- Nécessite l'utilisation de méthodes heuristiques pour pouvoir obtenir de bonnes partitions et pour l'optimiser.
- Comprise entre -1 et 1.
- Plus le résultat du calcul est proche de 1 plus le découpage est précis.
- Elle permet de comparer plusieurs partitions sur un même graphe .
- Fréquemment utilisée pour comparer les algorithmes de partitionnement d'un graphe donné.

2.4. Méthode d'optimisation de la modularité

Il existe plusieurs méthodes qui optimisent la modularité mais la majorité de ces méthodes sont appliquées à la détection de communautés disjointes.

- **Méthode Louvain**

Louvain [21]utilise une méthode d'optimisation de la modularité. Au départ chaque sommet est dans sa propre communauté, puis chaque sommet prend la communauté d'un de ces voisins de telle sorte que le gain de modularité soit maximal. Cette opération est répété plusieurs fois sur l'ensemble des sommets jusqu'à ce qu'aucun gain ne soit possible. Toute l'opération est ensuite répété sur le graphe des communautés.

- **FastGreedy [21]**

Comme Louvain cette méthode est basée sur une optimisation de la modularité. En partant du partitionnement le plus fin (une communauté par noeud), les communautés sont rassemblées progressivement si cela permet d'augmenter la modularité. Un des point fort de cette méthode est sa rapidité d'exécution.

3. Modularité chevauchantes

Le critère le plus largement admis pour qu'un ensemble de groupes forme une « bonne » décomposition du graphe est une densité intra-groupe élevée et une densité inter-groupes faible.

1. La modularité de Newman a été adaptée aux communautés chevauchantes
2. La mesure de qualité *MQ* de Mancoridis définie précédemment a été généralisée par Bourqui [29] pour une décomposition de l'ensemble des sommets d'un graphe en groupes chevauchants.

3.1. Adaptation de La modularité de Newman aux communautés chevauchantes

Nicosia et al[30]2009 ont mis en considération le problème d'extension de la modularité pour les communautés chevauchante, ils ont considéré le cas de réseau orienté mais non pondéré. ils ont commencé par l'expression générale suivante :

$$Q_{OV} = \frac{1}{m} \sum_{c=1}^{nc} \sum_{i,j} \left[r_{ijc} A_{ij} - s_{ijc} \frac{K_i^{out} K_j^{in}}{m} \right]$$

Ils ont fait une distinction entre les arrêts entrantes et sortantes (K_i^{out} K_j^{in}) dans les graphes orientés.

c :l'identifiant de la communauté.

$\mathbf{r_{ijc}}$, $\mathbf{s_{ijc}}$ expriment les contributions à la correspondance de somme de l'arrêt **i, j** dans le réseau et dans le modèle nul, dû aux adhésions multiples de **i** et **j** .

concernant les communautés chevauchantes les coefficients $\mathbf{r_{ijc}}$, $\mathbf{s_{ijc}}$ doivent dépendre des coefficients d'adhésion $\boldsymbol{\alpha_{i,c}}$, $\boldsymbol{\alpha_{j,c}}$.

Donc $\mathbf{r_{ijc}} = \boldsymbol{F(\alpha_{i,c}}$, $\boldsymbol{\alpha_{j,c})}$, F est une certaine fonction.

La limite $\mathbf{s_{ijc}}$ est liée au modèle nul de la modularité

$$\beta_{i\rightarrow,c}^{out} = \frac{\sum_j F(\alpha_{i,c} , \alpha_{j,c})}{n}$$

$$\beta_{i\leftarrow,c}^{in} = \frac{\sum_j F(\alpha_{j,c} , \alpha_{i,c})}{n}$$

ils ont fini par l'expression suivante :

$$Q_{OV} = \frac{1}{m} \sum_{c=1}^{nc} \sum_{i,j} \left[r_{ijc} A_{ij} - \frac{\beta_{i\rightarrow,c}^{out} K_i^{out} \beta_{j\leftarrow,c}^{in} K_j^{in}}{m} \right]$$

3.2. Modularité de Mancoridis adaptée aux communautés chevauchantes

Considérons un graphe $G = (V, E)$ où V est l'ensemble des sommets et E l'ensemble des arêtes. Soit une décomposition Cm en groupes chevauchants $Cm = \{Cm_1, Cm_2, \dots, Cm_k\}$ de l'ensemble des sommets V, telle que :

$\exists\ (i,j)\ tel\ que\ Cm_i \cap Cm_j \neq \emptyset\ \ et\ Cm = \cup\ \mathrm{Cm_i}$ (i = 1,..k).

- **Notation :**

$Cm_j \backslash Cm_i$ est le groupe Cmj dépourvu des sommets qu'il a en commun avec le groupe Cmi.

$|\mathrm{Cm_j}\backslash\mathrm{Cm_i})|$ (cardinal) : est le nombre de sommets du groupe $\mathrm{Cm_j}$ dépourvu des sommets qu'il a en commun avec $\mathrm{Cm_i}$

$\mathrm{E(Cm_i,\ Cm_j}\backslash\mathrm{Cm_i})$, représente le nombre d'arêtes entre le groupe $\mathrm{Cm_i}$ et le groupe $\mathrm{Cm_j}$ diminué des nœuds qui sont en communs entre $\mathrm{Cm_i}$ et $\mathrm{Cm_j}$.

La modularité Chevauchante (Recouvrante) de Mancoridis $\mathrm{MQ_{Over}}$ adaptée par Bourqui[29] $\mathrm{MQ_{Over}}$ est définie comme la différence entre la cohésion interne et la cohésion externe des groupes comme suit :

$$\mathbf{MQ_{Over} = MQ^{+} - MQ^{-}_{Over}} \qquad (1)$$

$\mathbf{MQB^{+}} = \frac{1}{\mathrm{k}}\sum_{\mathrm{i=1}}^{\mathrm{k}} \mathrm{S(Cm_i,\ Cm_i)}$ et $\mathbf{MBQ^{-}_{Over}} = \frac{1}{\mathrm{k(k-1)}}\sum_{\mathrm{i}}\sum_{\mathrm{j\neq i}} \mathrm{S_{Over}\ (Cm_i, Cm_j)}$

$S(C_i,\ C_j) = \frac{E(C_i, C_j)}{|C_i| * |C_j|}$. Tel que $E(C_i,\ C_j)$ est le nombre d'arêtes entre les sommets des groupes C_i et C_j. et $|C_i| * |C_j|$ représente le maximum d'arêtes qu'on peut avoir entre les deux groupes $\mathrm{C_i}$ et $\mathrm{C_j}$.

$$\mathbf{MQ_{Over}} = \frac{1}{k}\sum_{i=1}^{k} \frac{E(C_{mi}, Cm_i)}{|Cm_i| * |Cm_i|} - \frac{1}{k(k-1)}\sum_{i}\sum_{j\neq i} \frac{\mathrm{E(Cm_i, Cm_j}\backslash\mathrm{Cm_i})}{|\mathrm{Cm_i}| * |\mathrm{Cm_j}\backslash\mathrm{Cm_i}|} \qquad (2)$$

Remarque : la mesure $\mathrm{MQ_{Over}}$ est destinée aux graphes mono-partis.

3.3. Adaptation de la mesure MQ_{Over} à un graphe biparti

L'approche que nous devons implémenter est destinée aux réseaux bipartis, elle intègre une adaptation de la modularité de Mancoridis MQ_{Over} (définie précédemment) aux graphes bipartis. Nous présentons alors dans la section suivante cette adaptation.

Soit G (V, E) un graphe (réseau) biparti non orienté, non pondéré, V est l'ensemble des sommets (nœuds) et E l'ensemble des arêtes (liens), possédant deux classes de sommets disjointes ($V = V_r \cup V_b$ et $V_r \cap V_b = \emptyset$) telles qu'il n'existe aucune arête reliant deux sommets de la même classe.
Autrement dit, on a : $E \cap (V_r \times V_r) = \emptyset$ et $E \cap (V_b \times V_b) = \emptyset$.

Soit G (Vr, Vb, E) un graphe biparti non orienté, non pondéré. Vr et Vb représentent respectivement les ensembles de sommets rouges et bleus et E l'ensemble des arêtes. Soient Nr=|Vr|, Nb=|Vb|, Nr+Nb= n.

La mesure de qualité d'une décomposition des sommets en groupes chevauchants dans un graphe biparti sera : $\mathbf{MQB_{Over}} = \mathbf{MQB^{+}} - \mathbf{MBQ^{-}_{Over}}$ (3)

Soit une décomposition $Cm = \{Cm_1, Cm_2, \dots, Cm_k\}$ de l'ensemble des sommets Vr U Vb. On distinguera pour un groupe de sommets Cm_i, les deux sous ensembles des deux types de sommets par Cmr_i et Cmb_i avec $Cm_i = Cmr_i \cup Cmb_i$. Pour un groupe de sommets Cm_i le maximum d'arêtes qu'on peut avoir et $|Cmr_i| * |Cmb_i|$, c'est le cas d'un sous-graphe biparti complet, on a donc

$$\mathbf{MQB^{+}} = \frac{1}{k} \sum_{i=1}^{k} \frac{E(Cm_i, Cm_i)}{|Cmri| * |Cmbi|}$$

Pour MQ^{-}_{Over} on doit considérer le cas du sous-graphe complet contenant l'ensemble des sommets rouges et l'ensemble des sommets bleus des deux groupes de sommets Cm_i et Cm_j à la fois. Ce qui donne :

$$\mathbf{MQB^{-}_{Over}} = \frac{1}{k(k-1)} \sum_i \sum_{j \neq i} \frac{E(Cmi,\ Cmj|Cmi)}{|Cmr_i| * |Cmb_j \backslash Cmb_i| + |Cmb_i| * |Cmr_j \backslash Cmr_i|}$$

donc on a la modularité Overlapping : (4)

$$\mathbf{MQB_{Over}} = \frac{1}{k}\sum_{i=1}^{k}\frac{E(Cm_i, Cm_i)}{|Cmri|*|Cmbi|} - \frac{1}{k(k-1)}\sum_{i}\sum_{j\neq i}\frac{E(Cmi,\ Cmj|Cmi)}{|Cmr_i|*|Cmb_j\backslash Cmb_i| + |Cmb_i|*|Cmr_j\backslash Cmr_i|}$$

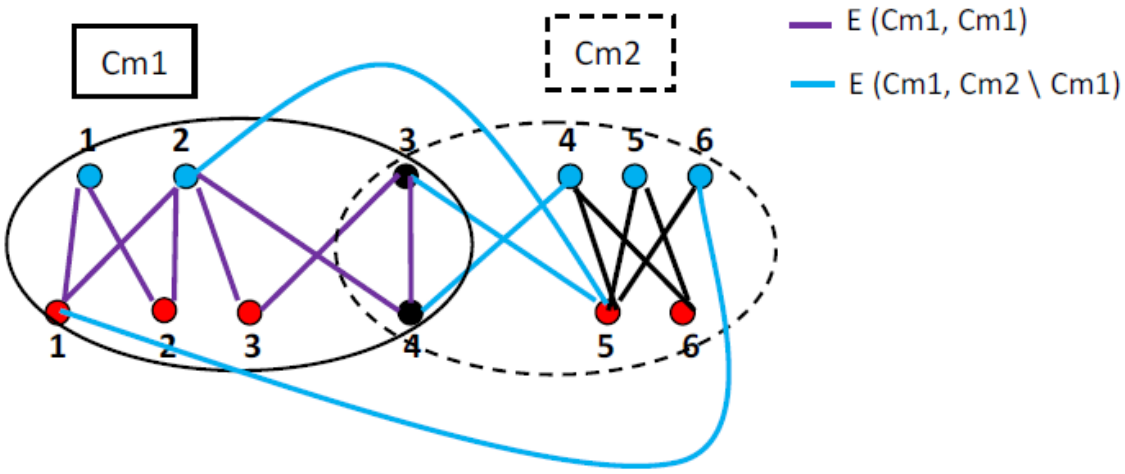


Figure 3.1 : Figure illustre certaines notions de la mesure MQB_{Over} sur deux communautés chevauchement.

$Cm_2\backslash Cm_1$: est le groupe Cm_2 dépourvu des sommets qu'il a en commun avec le groupe Cm_1 .

$E(Cm_1,\ Cm_1\backslash Cm_2)$, représente le nombre d'arêtes entre le groupe Cm_1 et le groupe Cm_2 diminué des nœuds qui sont en communs entre les deux groupes.

4. Optimisation de la modularité chevauchantes

L'optimisation de la modularité dans les communautés chevauchantes pose un problème, ça est dû à la difficulté de travailler et de comprendre ces communautés, à cause de ces problèmes il n'existe que très peu d'approches qui traitent cette catégorie.

4.1. Moyen d'optimiser la modularité chevauchantes

1. Applications des algorithmes de filtrages à des partitions pour optimiser la modularité.
2. Implémentation de certains algorithmes de détection de communautés chevauchantes qui maximisent la modularité comme l'algorithme « fusion ».

4.1.1. Algorithme fusion[18]

Tout au long de la procédure de fusion (introduite dans le chapitre précédent), la modularité des différents systèmes de classes varie, soit :

- De façon monotone décroissante (cliques maximales, arêtes),
- De façon croissante puis décroissante (cliques centrées).

La fusion de deux classes *Vi* et *Vj* entraîne leur suppression et l'apparition d'une nouvelle classe *Vi* ∪ *Vj* .La fusion de ces classes entraîne un *effet de chaîne*, c'est-à-dire le rattachement des éléments un à un à des classes de plus en plus grandes.

Deux modifications ont été apportées pour pallier à ces problèmes:

- ✓ Modification de la règle de choix : les deux classes réunies sont celles pour lesquelles la valeur *moyenne d*e variation de modularité est maximum. Ceci maximise la modularité ,favorise la fusion de petites classes et l'effet de chaîne a fortement baissé ;
- ✓ le processus de fusion est stoppé, pour ne pas aboutir à une seule classe, de surcroît de modularité faible. Des critères d'arrêt ont donc été introduits, comme de fixer le nombre de classes voulues.

4.1.2. Présentation de l'algorithme de filtrage à implémenter

Etape1 :

Un groupe constitué que de nœuds chevauchants n'apporte aucune connaissance, on doit donc supprimer les groupes obtenus ayant cette caractéristique. Cette suppression se fera en minimisant le nombre de liens entre groupes.
Cette minimisation se fera par un algorithme qui supprimera en premier lieu les groupes de nœuds chevauchants de petites tailles.

Tant qu'il ya des groupes constitués <u>exclusivement</u> de nœuds chevauchants

-Supprimer le groupe de nœuds ayant le plus petit cardinal ;

-Mettre à jour tous les autres groupes (le groupe supprimé n'existe plus.

Donc certains nœuds peuvent changer de statut de nœud chevauchant en nœud non chevauchant).

fait

Etape2 :

Après la fin de l'étape1 , les nœuds appartenant aux autres groupes vont être filtrés pour décider s'ils peuvent être considérés comme des nœuds chevauchants ,Soit *i* un nœud appartenant à plusieurs groupes et soit *E(i, Cmj)* le nombre de liens du nœud **i** avec le groupe Cm_j. On définit l'ensemble des groupes partageant le nœud i, comme $Ov_i = \{ Cmj \mid i \in Cmj \}$.

Si un nœud chevauchant appartient à un groupe, et s'il ne participe pas activement à la cohésion interne mais contribue beaucoup à la cohésion externe de ce groupe, on le supprimera de ce groupe pour augmenter la modularité.

Nous utiliserons à cet effet une équation que les nœuds chevauchants doivent vérifier pour appartenir à une communauté. Pour que le nœud i soit considéré comme appartenant à la communauté *Cmj*, la proportion de ses liens avec cette communauté doit être supérieure ou égale à la moyenne des proportions des liens de tous les groupes partageant ce nœud. Toutefois si le nœud i réalise un score d'au moins 0.5 il est considéré comme appartenant au groupe. Le nœud i doit donc vérifier l'une des conditions suivantes :

Si i ∈ *Cmrj (rouge)* alors $\frac{E(i,Cmj)}{|Cmbj|} \geq \boldsymbol{Min}\left(\left(\sum_j \frac{E(i,Cmj)}{|Cmbj|}\right)/ |\boldsymbol{Ov_i}|,\ \boldsymbol{o.5}\right)$

Si i ∈ *Cmbj* (bleu) alors $\frac{E(i,Cmj)}{|Cmrj|} \geq \boldsymbol{Min}\left(\left(\sum_j \frac{E(i,Cmj)}{|Cmrj|}\right)/ |\boldsymbol{Ov_i}|,\ \boldsymbol{o.5}\right)$

$|ov_i|$: représente le cardinal de ov_i.

Comme exemple on a le cas suivant:

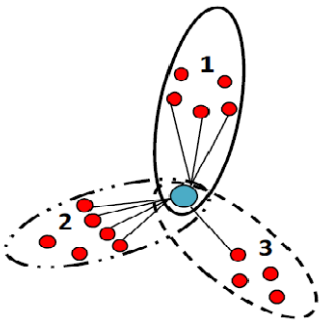


Figure 3.2 : Exemple de groupes pour lesquels on va appliquer l'étape 2.

Un nœud appartenant à 3 groupes mais n'appartient qu'à 2 communautés chevauchantes

Dans le schéma de la figure précédente, on vérifie les conditions suivantes:

pour la communauté 1, on a:

- 3/5 ≥ (3/5+4/6+1/4)/3, donc le nœud appartient à la communauté 1,
- pour la communauté 2, on a aussi 4/6 ≥(3/5+4/6+1/4)/3, donc le nœud appartient à la communauté 2,
- par contre pour la communauté 3, la relation n'est pas vérifiée

  ¼ <(3/5+4/6+1/4)/3, et 1/4 < 0.5, donc le nœud n'appartient pas à la communauté 3.

Etape3 :

Dernière étape de la méthode : supprimer les communautés ayant un seul nœud ou deux nœuds et placer chacun de ces nœuds dans les communautés du voisin qui a donné une meilleure modularité.

5. Conclusion

L'identification de communautés denses dans les grands réseaux d'interactions est un problème complexe notamment lorsqu'il s'agit de communautés chevauchantes.

La détection de ces communautés, revient à chercher une partition de l'ensemble des sommets qui maximise une fonction de qualité telle que la modularité. la maximisation de cette fonction est un problème NP-complet .

Quant il s'agit de communautés disjointes ça nécessite l'utilisation de méthodes heuristiques pour pouvoir obtenir de bonnes partitions. Mais dans le cas des communautés chevauchantes, la maximisation de la modularité est plus difficile.
Pour cela des études sont menées pour bien comprendre la notion de communautés chevauchantes et pour améliorer le partitionnement des réseaux.

Chapitre 4

Conception et implémentation

1. Introduction

La conception est une phase essentielle dans un projet, dans laquelle une réflexion globale est menée, dans un premier temps, sur la structure du travail et dans un second temps, sur les solutions techniques représentant la formalisation de l'idée.

Notre travail consiste à mettre en œuvre un algorithme de filtrage pour détecter des communautés chevauchantes sur un graphe biparti. Pour cela nous allons commencer par le codage des données à manipuler, à savoir le paramètre d'entrée, la représentation du graphe biparti, les structures de données employées. Nous présenterons ensuite les différents algorithmes implémentés pour réaliser l'application.

2. Codages des données

2.1. Format du graphe biparti

Dans un graphe biparti, il est impossible d'avoir un lien entre deux nœuds du même type.

Le graphe biparti est transformé en matrice (voir la figure), à cet effet nous avons représenté les nœuds de la première famille par des indices lignes, et les nœuds de la deuxième famille par des indices colonnes. les liens entre les nœuds sont représentés

par le contenu des cellules. Cela évite de gaspiller un espace mémoire inutilement et facilite certaines opérations effectuées pour détecter les communautés chevauchantes et leurs évaluations.

Par la suite cette matrice est mise dans un fichier texte dans un format particulier : C.à.d. qu'au lieu de mettre toute la matrice qui représente le graphe dans un fichier, on insère seulement le nombre de nœuds dans chaque famille et le couple de nœuds qui sont reliés sur une ligne du fichier d'entrée.

La représentation de ce graphe sur le fichier est comme suit:

- la première ligne du fichier contient deux valeurs :

  -la première indique le nombre de nœuds de la première famille (nombre de ligne)

  -la deuxième indique le nombre de nœuds de la deuxième famille (nombre de colonne).
- les lignes représentes les nœuds rouge.
- les colonnes représentes les nœuds bleus.
- les autres lignes contiennent les couples de nœuds qui se relient. Le nombre total de couples représentés sur le fichier, indique le nombre de liens existant dans le graphe.

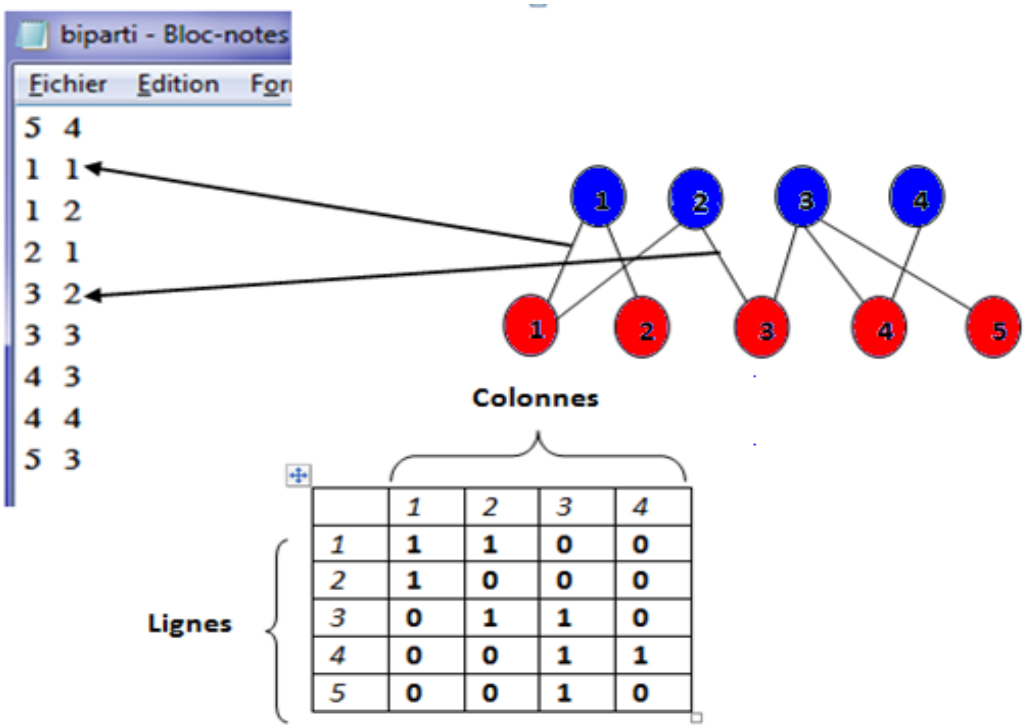


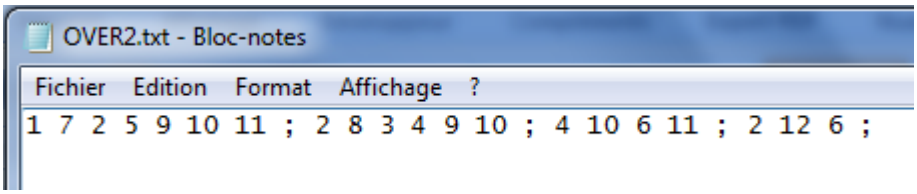
Figure4.1 : Capture d'écran d'un fichier sur lequel on a sauvegardé un graphe biparti ainsi que sa représentation matricielle.

2.2. Format des communautés chevauchantes

La décomposition des nœuds en groupes chevauchants est obtenue par une méthode évolutionnaire. Cette méthode a fait l'objet d'un sujet de master [23]. Ces groupes sont récupérés dans un fichier texte dont les communautés sont séparées par un délimiteur le « ; ». A ces groupes de nœuds chevauchants sera appliqué l'algorithme de filtrage de nœuds.

Exemple:

Dans l'exemple qui suit nous avons 4 groupes chevauchants :



Groupe1 : 1 7 2 5 9 10 11

Groupe 2 : 2 8 3 4 9 10

Groupe 3 : 4 10 6 11

Groupe 4 : 2 12 6

Figure4.2 : Capture d'écran d'un fichier sur lequel on a sauvegardé les groupes

2.3. Les structures de données

Communauté

Communauté est un objet composé de deux éléments : le premier est un entier pour indiquer l'IdCom (identifiant de communauté) pour chaque communauté et le deuxième est un tableau dynamique de type entier (ListNœud) qui dispose des nœuds appartenant à cette communauté. Un nœud peut appartenir à une ou plusieurs communautés, ça veut dire qu'on peut trouver le même nœud dans plusieurs ListNœud (il s'agit des communautés chevauchantes).

IdCom	ListNoeud

Groupe de communautés Chevauchantes :

Est un tableau dynamique qui contient l'ensemble des communautés (précédentes).

Communauté1	Communauté2	. . . . .	Communauté k

2.4. Paramètres

Pour réaliser le filtrage on a besoin d'un paramètre d'entrée du programme « **Nomfichier »** c' est une variable de type chaine de caractère utilisée pour désigner le nom du fichier (.txt) sur lequel les communautés chevauchantes sont sauvegardés.

3. L'organigramme de l'application

L'organigramme ci-dessous représente le schéma général de l'application. Il comporte 3 parties :

(1) Préparation: lire le paramètre d'entrée ,le graphe biparti puis générer sa matrice.

(2) Exécution de l'algorithme de filtrage.

(3) Affichage des résultats obtenus.

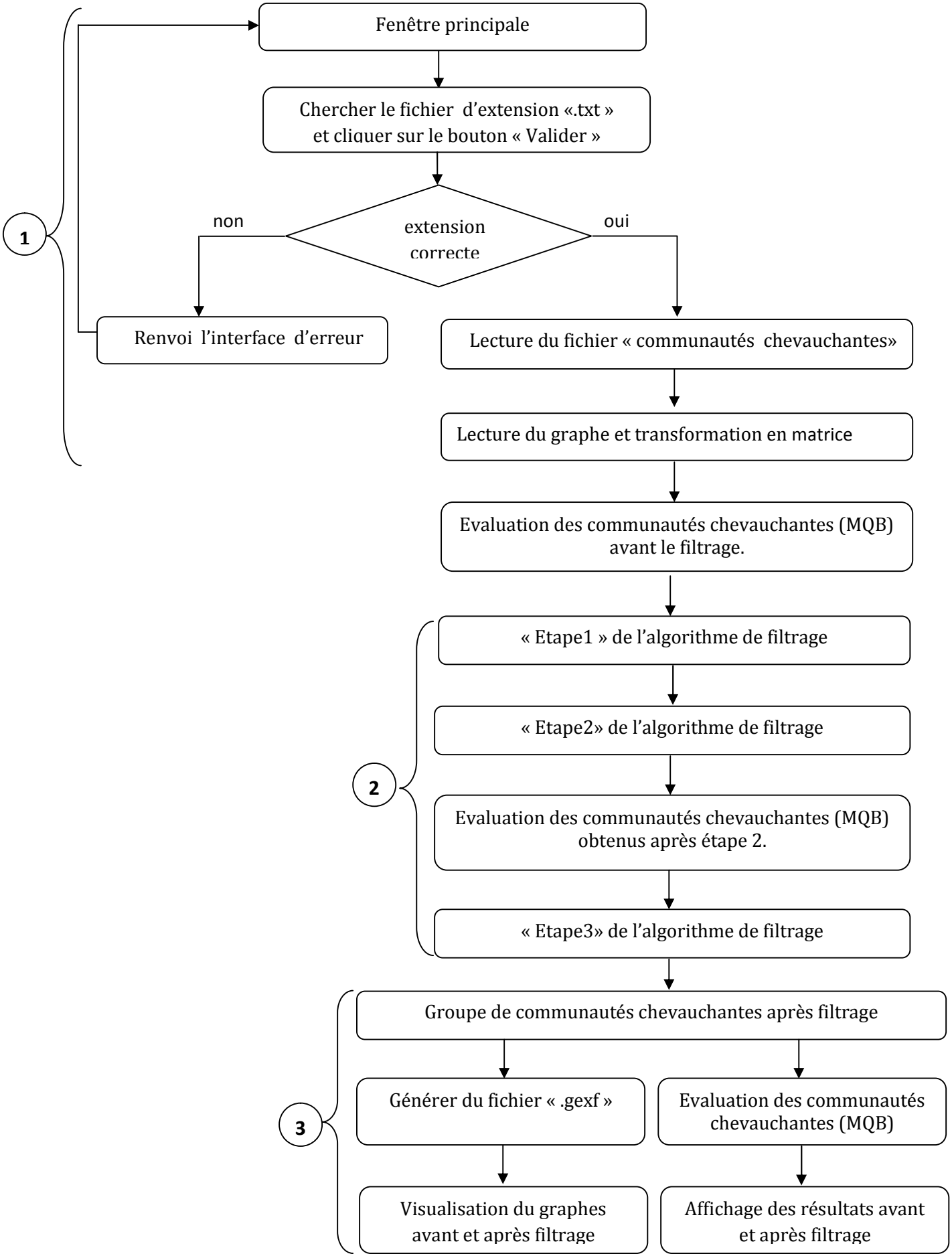


Figure4.3 : Organigramme de l'application

4. Les algorithmes

Après la présentation des structures de données employées afin de coder les variables indispensables qui sont employées dans l'algorithme et le schéma générale de l'application, on va aborder dans cette partie les différents algorithmes implémentés pour réaliser ce travail.

1. Lecture des communautés chevauchantes et graphe biparti

1.1. La lecture du fichier d'entrée

On fait la lecture du fichier qui contient les communautés chevauchantes , puis on va placer chacune d'elle dans un tableau dynamique ayant des objets : de type communauté expliqué dans la partie (2.3. Les structures de données).

Algorithme1

```
début
  • ouvrir en écriture le fichier (nomfichier)
  • créer un nouveau tableau
    dynamique(groupecommunauteChevauchante) qui contient des objet
    de type communaute.
  • communaute1←une nouvelle communauté
  • lire la première ligne
  • nœuds ←nombres de token
  • cpt←0

pour  j allons de 1 à nœuds faire
      début
            • communaute1.IdCom ←cpt
            • token← le prochain token
            • si (token != ';') alors
              ajouter le token à communauté1.ListNoeud
              si non
               ajouter communauté1 à groupecommunauteChevauchante
            • cp++
            • communauté1←une nouvelle communauté
          fin
fin
```

1.2. Lecture du graphe biparti

Dans cette procédure on va lire le fichier qui contient le graphe biparti puis on va le transformer en matrice.

Algorithme2

```
début
  • ouvrir en écriture le fichier (nomfichier)
  • lire la première ligne
  • nbrNoeudBleu← le premier élément de la ligne
  • nbrNoeudRouge← le deuxième élément de la ligne
 tant que (ce n'est pas la fin du fichier) faire
    début
                lire la ligne suivante
                ligne← le premier élément de la ligne
                colonne ← le deuxième élément de la ligne
          matrice[ligne][colonne] ←1
                  la ligne suivante
    fin
fin
```

2. Les étapes de l'algorithme de filtrage

2.1. Etape1

Dans cette étape on va supprimer les communautés exclusivement chevauchantes et on va mettre à jour tous les nœuds.
l'algorithme ci-dessous explique le déroulement de l'étape1.

Algorithme 3: étape1

```
début
  • LectureFichierEntrée ()
  • rechercheCommunauteExclusivementChevauchante()
  • nbreOcNoeudChev()
```

```
pour chaque communaute1 du groupcomExclChev  faire
début
```

- *calculer la communaute ayant* la plus petite *taille(peu de nœuds)*
- ident← *communaute1. IdCom*

```
pour chaque communaute2 du groupcomExclChev  faire
  début
    Si (communaute2.IdCom = ident) alors
      début
         pour chaque nœud1 de communauté2 faire
          début
              pour chaque liste du NbrOccNoeudChev  faire
               début
                  Si (communaute2.noeud1 = liste.noeud) alors
                 début
```

- *liste.cpt=liste.cpt-1;*

```
                si(liste.cpt=1) alors
```

- *//le nœud1 n'est plus chevauchant*
- *supprimer liste du NbrOccNoeudChev*

```
              fin
          fin
      fin
```

- *supprimer communaute1 du groupcomExclChev*
- *supprimer communaute1* du *groupeCommunauteChevauchant*

```
   fin
  fin
 fin
fin
```

Cette étape contient un ensemble de procédures :

2.1.1. Rechercher les nœuds chevauchants dans chaque communauté

On va parcourir chaque communauté pour chercher les nœuds chevauchants.

Algorithme4

```
début
```

- *LectureFichierEntrée ()*
- *créer un nouveau tableau dynamique(groupcomChev)*

pour chaque communaute1 du groupeCommunauteChevauchant faire
début

- *créer une nouvelle communaute(comChevauchante)*

pour chaque communaute2 du groupeCommunauteChevauchant faire
début
Si (communaute1.IdCom != communaute2. IdCom) alors
début

- *comChevauchante.IdCom ← communaute1.IdCom*

pour chaque nœud1 de communauté 1 faire
début
pour chaque nœud2 de communauté 2 faire
début
si (nœud1= noeud2) et (nœud1∉ comChevauchante.ListNoeud)alors

- *comChevauchante.ListNoeud ← noeud1*

fin
fin
fin
fin

- *ajouter comChevauchante à groupcomChev.*

fin
fin

2.1.2. Rechercher les communautés exclusivement chevauchantes

On va parcourir toutes les communautés pour extraire celles qui sont exclusivement chevauchantes.

Algorithme5

---

début

- *rechercheNoeudChevauchants()*
- *creer un nouveau tableau dynamique(groupcomExclChev)*

pour chaque communaute 1du groupeCommunauteChevauchant faire
début
pour chaque communaute2 du groupcomChev faire
début
Si (communaute1.IdCom = communaute2. IdCom) alors
début
si (la taille de communaute1=taille de communauté2) alors

- *ajouter communaute1 à groupcomExclChev*

```
      fin
    fin
  fin
fin
```

2.1.3. Calculer le nombre d'occurrence de chaque nœud chevauchant

Cette procédure est utilisée pour la mise à jour des nœuds parce que certains nœuds peuvent changer de statut de nœud chevauchant en nœud non chevauchant. cette procédure est utilisée dans l'étape2.

Algorithme6

```
début
• rechercheNoeudChevauchants()
• créer un tableau dynamique (NbrOccNoeudChev)d'objet
  (contient l'identifiant du nœud et son nombre d'occurrence)
• cpt← 0
  pour chaque communaute1 du groupcomChev faire
  début
    • liste ← novelle occurrence d'objet
    pour chaque nœud1 de communauté 1faire
    début
      • liste.noeud= nœud1
      pour chaque communaute2 du groupcomChev faire
        début
          pour chaque nœud2 de communauté2 faire
            début
            si (nœud1= noeud2) et (nœud1 ∉ NbrOccNoeudChev) alors
              • cpt← cpt+1;
            fin
        fin
          • liste.cpt← cpt
          • cpt← 0
          • ajouter liste à NbrOccNoeudChev
      fin
  fin
fin
```

2.2. Etape2

Après la fin de l'étape1 , les nœuds appartenant aux autres groupes vont être filtrés pour décider s'ils peuvent être considérés comme des nœuds chevauchants.

Algorithme 7: étape2

```
début
  • étape1()
  • somme()
pour chaque communaute1 du groupeCommunauteChevauchant faire
 début
   pour chaque tab du tabResultat faire
   début
     pour chaque tab2 du tabSomme faire
         début
              Si (tab2 . noeudChev = tab.noeudChev) et
              (communaute1.IdCom =tab.IdCom) alors
               début
                si (  tab.resultat <minimum (somme,0.5) alors
                si noeudChev  appartient à la communaute1 alors
                • supprimer le
               fin
         fin
    fin
  fin
fin
```

Pour réaliser cette étape on a utilisé un ensemble de procédures :

2.2.1. Le nombre de lien de chaque nœud chevauchant avec chaque communauté

Algorithme 8

```
début
  • étape1()
  • LectureGrapheBiparti ()
  •   nbreOccNoeudChev()
```

- *créer un tableau dynamique (tabResultat)ayant des objets de type resultat et IdCom et le noeudChev*

pour *chaque communaute1 du groupeCommunauteChevauchant* ***faire***
début

- *nbreLien ←0*
- *sauvegarder l'identifiant de communaute dans le tableau*
- *chercher le nombre de nœuds rouge dans la communaté1(nbNoeudRouge)*
- *chercher le nombre de nœuds bleu dans la communaté1(nbNoeudBleu)*

pour *chaque liste du NbrOccNoeudChev* ***faire***
début

- *nœudChev← liste.noeud*
- *sauvegarder le nœudChev dans le tableau*

si *nœudChev est bleu* ***alors***
début
pour chaque nœud1 de communauté 1faire
début
si il ya un lien entre nœudChev et nœud1 ***alors***

- *nbreLien ← nbreLien****++***

fin

- *resultat ← nbrLien/nbNoeudBleu*

fin
si *nœudChev est rouge* ***alors***
début
pour chaque nœud1 de communauté 1faire
début
si il ya un lien entre *nœudChev et nœud1 alors*

- *nbreLien ← nbreLien****++***

fin

- *resultat ← nbrLien/nbNoeudRouge*
- *sauvegarder le resultat dans le tableau*

fin
fin
fin

2.2.2. La somme des liens de chaque nœud chevauchant avec toutes les communautés

Algorithme9

```
début
  • nbreLienAvecCom()
  • créer un tableau dynamique (tabSomme)ayant des objets de type
    somme et NoeudChev
 pour chaque liste du NbrOccNoeudChev faire
   début
    • somme←0
     pour chaque tab du tabResultat faire
        début
          Si (liste.noeud = tab.noeudChev) alors
          début
            • somme←somme+(resultat/ liste.cpt)
            • ajouter noeudChev au tabSomme
          fin
       fin
    • ajouter somme au tabSomme
  fin
fin
```

2.3. Etape 3

Dernière étape de la méthode : supprimer les communautés ayant un seul nœud ou deux nœuds et placer chacun de ces nœuds dans les communautés du voisin qui a donné une meilleure modularité .

Algorithme10:étape3

```
début
  • étape2 ()
  • créer un tableau dynamique (tabMQBover)ayant des objets de type
    MQBover et IdCom, IdComSup
pour chaque communaute1 du groupeCommunauteChevauchant faire
```

```
début
  Si (taille communaute1 <= 2) alors
    début
```

- *ajouter communaute1.IdCom au tableau*

```
    pour chaque nœud1 de communauté 1faire
       début
       pour chaque communaute2 du groupeCommunauteChevauchant faire
          début
            Si (communaute1.IdCom != communaute2.IdCom) alors
             début
              pour chaque nœud2 de communauté2 faire
                 début
                  si il ya un lien entre nœud2 et nœud1 alors
                   début
```

- *ajouter communaute2.IdCom au tableau*
- *ajouter noeud1 au tableau*
- *placer le noeud1 dans la communaute2*
- *calculMQBover()*
- *ajouter MQBover dans le tableau*

```
                  fin
                fin
              fin
            fin
          fin
      fin
  fin
pour chaque communaute1 du groupeCommunauteChevauchant faire
 début
     pour chaque tab du tabMQBover faire
      début
        pour chaque nœud1 de communauté 1faire
         début
         Si (tab.IdComSup = communaute1. IdCom)et (tab.noeud=noeud1)
          début
```

- *chercher dans le tableau (tabMQBover) la communaute qui a donnée la meilleur modularité*
- *placer noeud1 dans cette communauté.*

```
          fin
        fin
      fin
    fin
```

- *supprimer communaute1 du groupeCommunauteChevauchant*

```
fin
```

3.1. Evaluation de la qualité des communautés chevauchantes

Pour évaluer la qualité du partitionnement en communautés chevauchantes, nous utilisons la métrique MQBover présenté dans le chapitre précédant. A cet effet, nous avons programmé la procédure dont : « Algorithme ».
Nous donnons d'abord des explications sur les variables utilisées et nous enchainons sur la description de l'algorithme.

taille: la taille du groupe de communautés chevauchantes

nbrLienInterne : est le nombre de liens interne pour une communauté quelconque

nbrLienExterne : est le nombre de liens entre deux communautés, c.à.d. que chaque lien doit avoir obligatoirement une extrémité dans la première communauté et l'autre extrémité dans la deuxième communauté.

nbrRouge : indique le nombre de nœuds de la première famille.

nbrBleu : indique le nombre de nœuds de la deuxième famille.

Algorithme11

```
début
  • lectureGrapheBiparti()
  • Initialiser MQBover,MQBplus, MQBmoins à zéro
// MQBplus
pour(i allons de 0 jusqu'à taille-1) faire
 début
   • initialiser nbrBleu, nbrRouge à zéro.
   • initialiser nbrLienInterne à zéro.
   pour chaque communaute du groupeCommunauteChevaauchant faire
     début
        pour chaque nœud de cette communauté faire
          début
            si le nœud appartient aux nœuds bleus alors
              • nbrBleu++
            si non
              • nbrRouge++
      pour chaque voisin du nœud faire
```

si *le voisin appartient à la même communauté que nœud* **alors**

- *nbrLienInterne++*

fin

- *nbrLienInterne← nbrLienInterne/2*

fin

- *MQBplus← MQBplus+(nbrLienInterne/ nbrBleu* nbrRouge)*

fin

- *QBplus← MQBplus/nbre de communauté*

// MQBmoins

- *initialiser nbrBleu1,nbrRouge1, nbrBleu2,nbrRouge2 à zéro.*
- *initialiser nbrNoeudCommunBleu ,nbrNoeudCommunRouge, nbrNoeudCommun à zéro.*

pour *chaque communaute1 du groupeCommunauteChevaauchant* **faire**
début
pour *chaque communaute2 du groupeCommunauteChevaauchant* **faire**
début
Si *(communaute1.IdCom != communaute2. IdCom)* **alors**
début

- *initialiser nbrLienExterne à zéro*

pour *chaque nœud1 de communauté 1* **faire**
début
pour *chaque nœud2 de communauté 2* **faire**
début
si *(nœud1= noeud2)* **alors**
début

- *nbrNoeudCommun++*

si nœuds 1 est rouge alors

- *nbrNoeudCommunRouge++*

si non

- *nbrNoeudCommunBleu++*

fin
fin
si *noeud1 est rouge* **alors**
début

- *nbrRouge1++*

pour *chaque nœud2 de communauté 2* **faire**
si *il ya un lien entre noeud1 et noeud2* **alors**

- *nbrArcExterne++*

fin
si non
début

- *nbrBleu1++*

pour *chaque nœud2 de communauté 2* ***faire***
si *il ya un lien entre noeud1 et noeud2* ***alors***

- *nbrArcExterne++*

fin
fin
pour *chaque nœud2 de communauté 2* ***faire***
début
si noeud2 est rouge

- *nbrRouge2++*

si non

- *nbrBleu2++*

fin

- *MQBmoins ← MQBmoins+((nbrArcExternenbrNoeudCommun)/ ((nbrRouge1*(nbrBleu2-nbrNoeudCommunbleu))+(nbrBleu1*(nbrRouge2-nbrNoeudCommunRouge))))*
- *initialiser nbrBleu1,nbrRouge1, nbrBleu2,nbrRouge2 à zéro.*
- *initialiser nbrNoeudCommunBleu ,nbrNoeudCommunRouge, nbrNoeudCommun à zéro*

fin
fin
fin

- *MQBmoins ← MQBmoins/(nombrecommunauteChevauchante* (nombrecommunauteChevauchante -1))*
- *MQBover←MQBplus-MQBmoins;*

fin

3.2. Génération du fichier.gexf

En plus de l'affichage des résultats du partitionnement, il est plus pratique de visualiser graphiquement les communautés résultantes, à cet effet, nous avons adopté un logiciel de visualisation qui reçoit en entré un *fichier.gexf* au forma *xml* pour représenter le graphe. Ce fichier se compose de deux parties :

1. l'entête,
2. le corps.

L'entête représente la partie commune de tous les documents GEXF, dans lequel est définit la version du XML et le type de codage employé, ainsi que l'élément racine du document gexf ; qui appartient à l'espace de nom : *http://www.gexf.net/1.1draft.*

Le corps est une partie obligatoire, qui consiste à définir la topologie du graphe, on déclare donc dans cette partie les balises suivantes : graphe, node, edge;

Ce type de fichier possède plusieurs avantages: en plus de sa simplicité, il permet de définir le type de graphe employé: orientés, non-orientés, statique ou dynamique ; d'attribuer nom, couleur, taille, position à chacun des nœuds et d'attribuer nom, couleur, poids, pour chacune des arêtes.

Algorithme12

début

- *ouvrir en écriture(fichier," grapheComChev.gexf ")*
- *écrire dans le fichier("<?xml version=\"1.0\"encoding=\"UTF8\"?>")*
- *écrire dans le fichier ("<gexf xmlns =\"http://www.gexf.net/1.2draft\"xmlns:viz =\"http://www.gexf.net/1.2draft/viz\">")*
- *écrire dans le fichier ("\t<graph defaultedgetype = \"undirected\">")*
- *écrire dans le fichier ("\t\t<nodes>")*

pour *chaque communaute1 du groupeCommunauteChevauchant* ***faire***
début
pour *chaque nœud de communaute1* ***faire***
début
si *le nœud est chevauchant* ***alors***

début

- *appliquer la couleur gris*
- *écrire dans le fichier(les communautés dont il appartient/son identifiant)*

fin
si *le nœud est rouge* ***alors***

début

- *écrire dans le fichier ("\t\t\t<nodeid=\""+communaute.IdCom+","+noeud+"\" label=\""+ communaute.IdCom +","+ noeud +"\">");*
- *appliquer la couleur rouge*

fin

si non

début

- *écrire dans le fichier ("\t\t\t<node id=\""+communaute.IdCom+"/"+noeud+"\" label=\""+ communaute.IdCom +"/"+ noeud +"\">");*
- *appliquer la couleur bleu*

fin

fin

fin

- *écrire dans le fichier ("\t\t</nodes>");//noeud*
- *écrire dans le fichier ("\t\t<edges>");//arret*

- associer pour chaque arête du graphe ses deux nœuds d'extrémités
- arête←0

début

- *écrire dans le fichier* ("\t\t\t<edge id=\""+ arête +"\" source=\""+noeudSource+ "\" target=\""+noudDestination +"\"></edge>")
- arête++

fin

- *écrire dans le fichier ("\t\t</edges>");*
- *écrire dans le fichier ("\t</graph>");*
- *écrire dans le fichier ("</gexf>");*
- *fermer le fichier*

fin

5. Conclusion

Après avoir présenté les différentes structures de données employées et le paramètre d'entrée de l'algorithme ainsi que l'architecture générale schématisé avec un organigramme et les différents algorithmes employés dans le but de réaliser l'application. on va présenter dans le chapitre suivant en plus, de l'environnement de développement, les résultats des tests effectués sur l'algorithme de filtrages afin d'évaluer ses performances.

Chapitre 5

Mise en œuvre et expérimentation

1. Introduction

Apres avoir défini les différentes étapes de l'algorithme de filtrage que nous allons implémenter pour les communautés chevauchantes. Nous présenterons dans ce chapitre la mise en œuvre de cet algorithme.

La première section décrit la plate forme de développement utilisée. Les sections suivantes portent sur l'architecture générale de l'application ainsi que les résultats expérimentaux que nous avons obtenus.

2. Environnement de développement

Pour implémenter l'algorithme de filtrage vu dans le chapitre précédant. Nous avons opté pour le langage java comme outil de développement dans un environnement Windows, et les résultats de filtrage des partitions obtenus par notre application sont visualisés à l'aide du logiciel Gephi (Gephi 0.7 alpha).

2.1. Environnement java

Java est un langage de programmation, développé par « Sun Microsystems Inc. », est disponible pour les principales plates formes du marché, qu'il s'agit de l'Unix, Windows, ou autres et est totalement gratuit.

Java est un langage de programmation à usage général, évolué et orienté objet dont la syntaxe est proche du C.

Java possède de nombreuses caractéristiques :

- **interprétable :** La source est compilé en pseudo code ou byte code puis exécuté par un interpréteur Java : une machine virtuelle JAVA (JVM). Ce concept est à la base du slogan de Sun pour Java : WORA (Write Once, Run Anywhere : écrire une fois, exécuter partout).

- **Portable :** Il est indépendant de toute plate-forme. Il n'y a pas de compilation spécifique pour chaque plate forme. Le code reste indépendant de la machine sur laquelle il s'exécute. Il est possible d'exécuter des programmes Java sur tous les environnements qui possèdent une machine virtuelle JAVA. Cette indépendance est assurée au niveau du code source grâce à Unicode et au niveau du byte code.

- **Orienté objet :** comme la plupart des langages récents, Java est orienté objet. Chaque fichier source contient la définition d'une ou plusieurs classes qui sont utilisées les unes avec les autres pour former une application. Java n'est pas complètement objet car il définit des types primitifs (entier, caractère, flottant, booléen,...).

- **Simple** : le fait que sa syntaxe soit basée sur celle de C++, mais dépouillée de tous les mécanismes complexes, redondants et inutiles.

- **Performant et rapide** : En effet, Java est d'une rapidité extraordinaire grâce à ses compilateurs spéciaux. Plus qu'un langage puissant, Java est une plate- forme de développement comportant une bibliothèque de classes très riches et de nombreux outils et interfaces de programmations applicatifs (API).

- **Fortement typé :** toutes les variables sont typées et il n'existe pas de conversion automatique qui risquerait une perte de données. Si une telle conversion doit être réalisée, le développeur doit obligatoirement utiliser un cast ou une méthode statique fournie en standard pour la réaliser.

- **Assure la gestion de la mémoire :** l'allocation de la mémoire pour un objet est automatique à sa création et Java récupère automatiquement la mémoire inutilisée

grâce au garbage collector qui restitue les zones de mémoire laissées libres suite à la destruction des objets.

- **Gestion et récupération des erreurs d'exécution (exceptions) :** Comme la plupart des langages modernes, Java dispose de mécanismes permettant de récupérer les erreurs d'exécution et de reprendre la main. Ce qui permet d'éviter les Bugs des logiciels.

- **Gratuit** : Sun fournit gratuitement la machine virtuelle Java avec les outils de développement de base et certains outils plus évolués. D'autres outils professionnels payants qui bénéficient de support technique plus développés sont mis à la disposition des développeurs.

- **Exécution sécurisée** : Les programmes s'exécutent sur une machine virtuelle permettant de protéger le système d'exploitation et les autres programmes utilisateurs des codes malveillants.

- **Bibliothèque riche**: Des composants fournis en standard avec Java couvrent de nombreux domaines; interface utilisateur graphique, accès aux bases de données, accès aux fichiers, accès au réseau, …etc.

2.2. Environnement gephi

Gephi est l'un des logiciels les plus employés à l'heure actuelle pour visualiser, analyser et explorer en temps réel des graphes de tout type. Il est écrit en Java et basé sur la plateforme Netbeans. Il se distingue par sa facilité d'utilisation. Il propose différents algorithmes de spatialisation paramétrables et permet également de travailler l'aspect du graphe (couleur et taille des nœuds et des arêtes), manuellement ou en fonction de données numériques.

Quelques fonctionnalités de gephi:

- Visualisation
- Algorithmes de placement (**layout**) efficaces.
- Métriques de réseaux : centralités, densité, diamètre, détection de communautés.

- Tableur adapté aux données de réseaux pour combiner visualisation et analyse statistique.
- Filtrage.
- Support des réseaux dynamiques et des graphes hiérarchiques .

Les deux premières fonctionnalités correspondent à nos besoins :

- **Visualisation :**

Permet la visualisation des communautés et la manipulation des différents nœuds de chacune d'elle, le schéma suivant illustre les divers éléments de l'interface de visualisation .

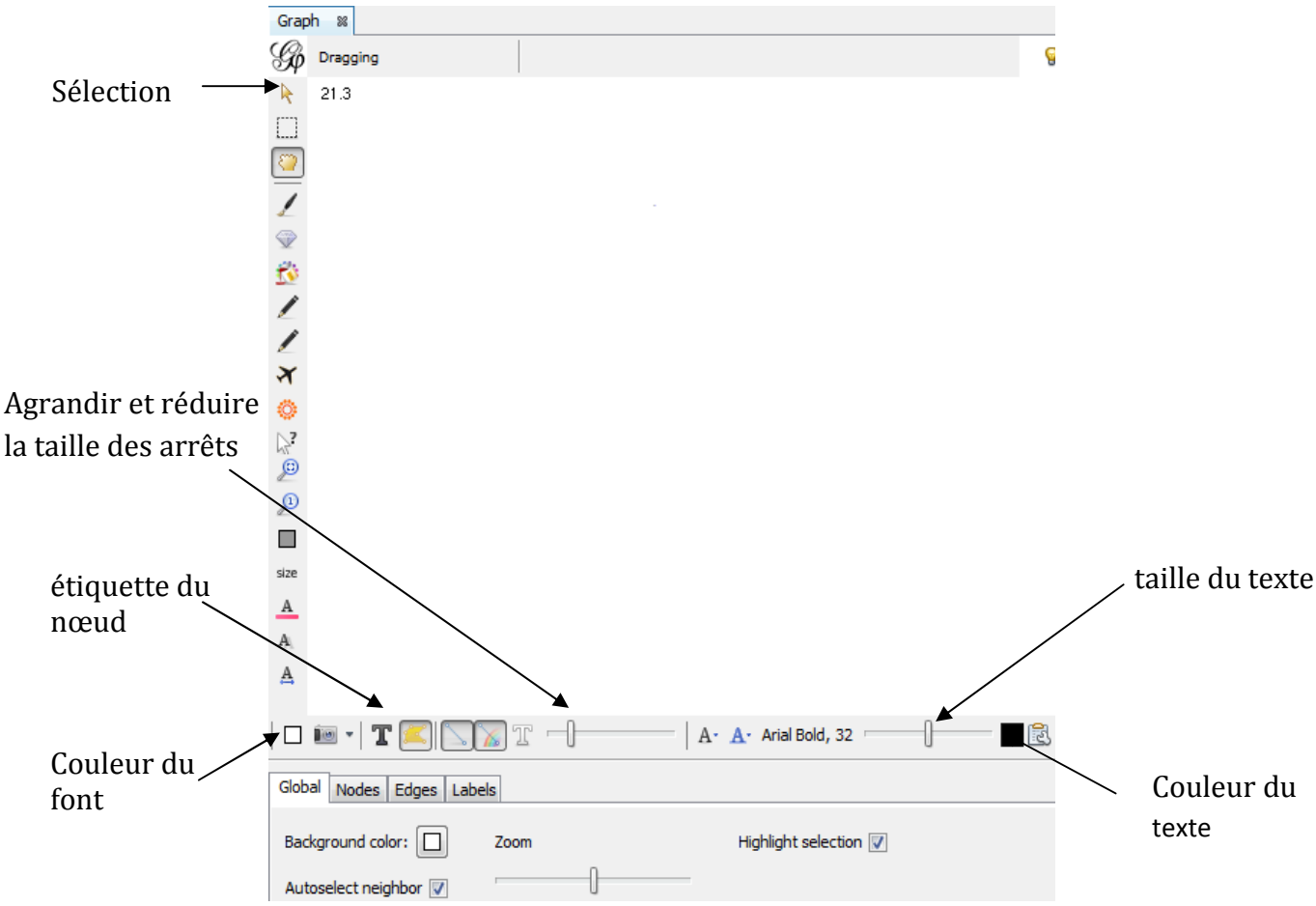


Figure5.1 : Vue générale de l'interface de visualisation de Gephi

➢ **Layout**

On a choisi d'appliquer l'algorithme *Force Atlas* car il permet de rapprocher les nœuds de même taille ,lors de la programmation. Les nœuds d'une même communauté doivent avoir la même taille.

La procédure est décrite comme suit (illustrer dans le schéma suivant):

1- Cocher la case « *Atraction Distrib* ».

2- Cocher la case « *Ajust by Sizes* ».

3- Cliquer sur « *Run* ».

4- Lorsque le graph s'est stabilisé, cliquer sur « *Stop*».

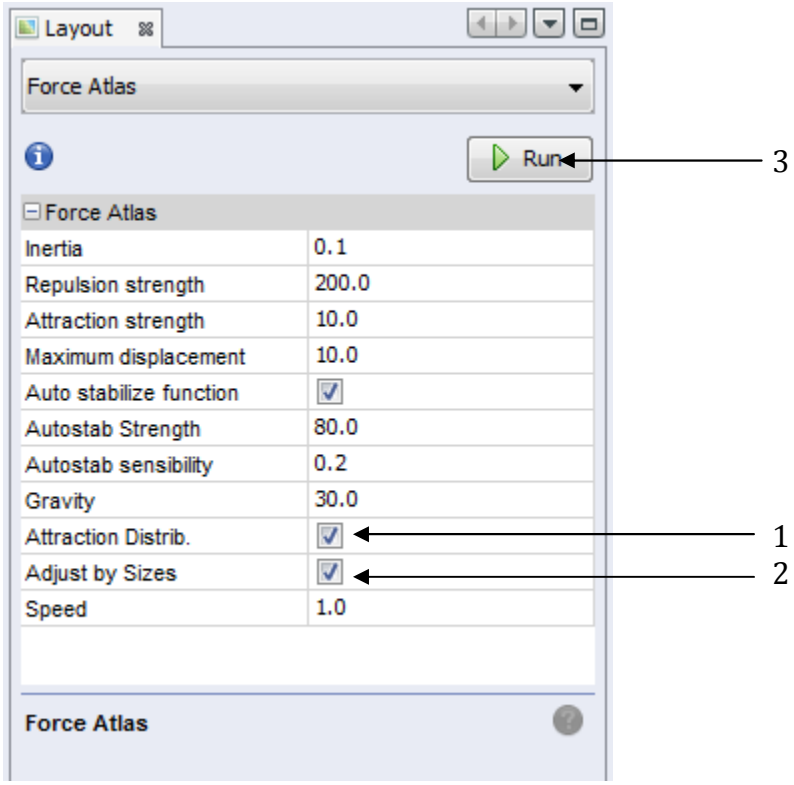


Figure5.2: Vue générale de l'interface de Layout

➢ **Fichier d'entrée/sortie de Gephi**

- *Fichiers d'entré des données* : CSV, GDF, GEXF, GML, GraphML, Graphviz DOT, Pajek NET, Tulip TLP, Ucinet DL, XGMML.
- *Fichiers de sortie des données* : CSV, GDF, GEXF, GraphML.
- *Fichiers de sorties graphiques* : PDF, SVG, PNG.

3. Description des interfaces

3.1. La fenêtre principale

Cette interface contient le chemin du fichier ayant les communautés chevauchantes sur lesquelles on va appliquer l'algorithme de filtrage.

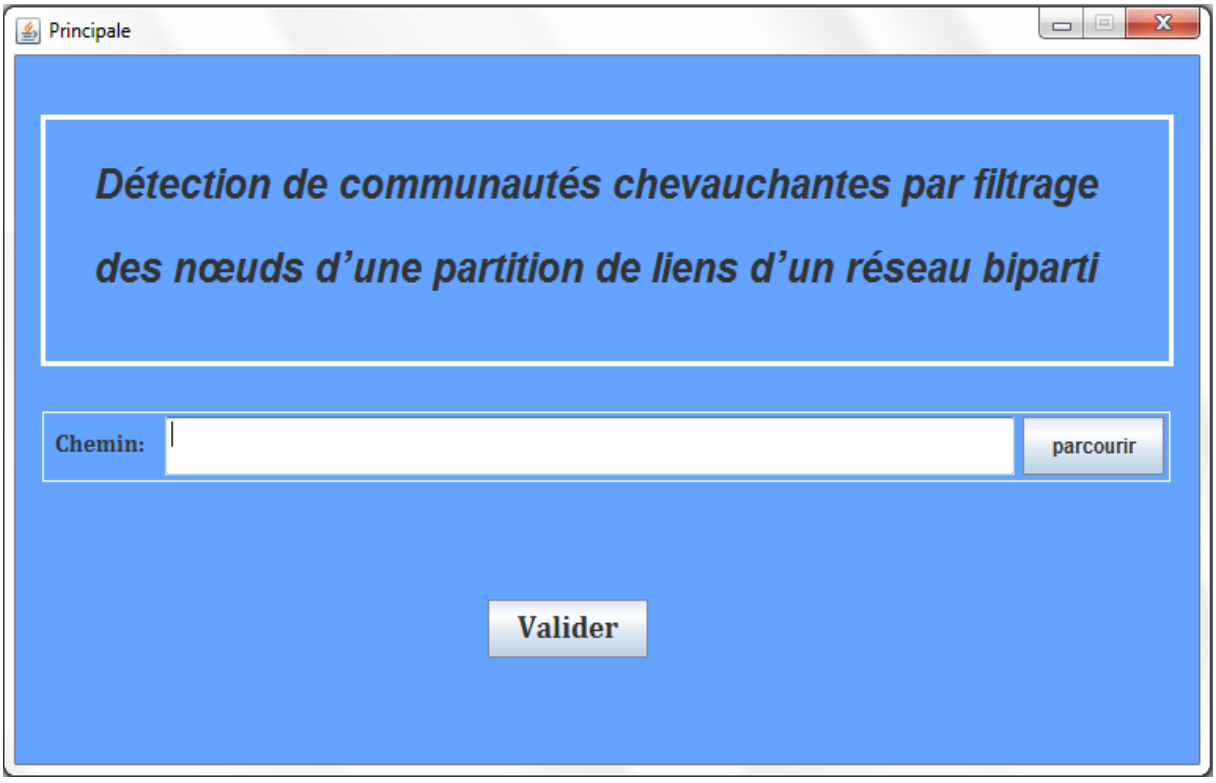


Figure5.3 : La fenêtre principale

➢ pour charger le fichier cliquer sur le bouton « parcourir» , l'interface suivante va apparaitre.

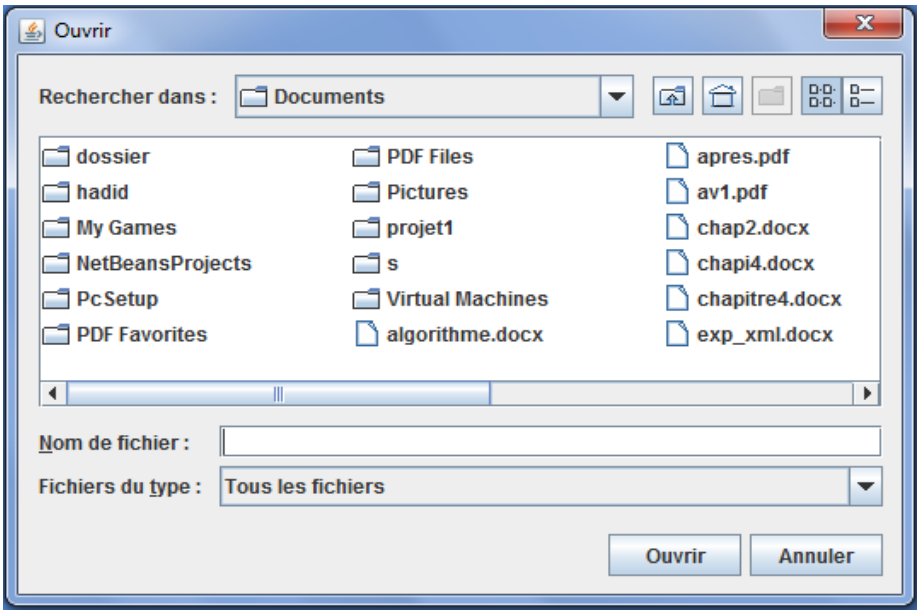


Figure5.4 : Fenêtre de chargement de fichier

- Après le chargement du chemin cliquer sur le bouton « Valider » deux positions sont possibles.

1. Si le fichier d'entré est correcte l'interface suivante va apparaitre, elle contient les résultats de l'algorithme de filtrage.

   Les résultats retournés sont :

   - Nombre de communautés avant et après le filtrage.(Nbre_Com_avant , Nbre_Com_Après)
   - Affichage de toutes les communautés avec les nœuds qui les composent après le filtrage.
   - Evaluation de qualité de partition avant et après filtrage(MQBover_Avant , MQBover_Après)
   - Le bouton « **Visualisation avant le filtrage** » permet de visualiser les communautés initiales à l'aide du logiciel gephi.
   - Le bouton « **Visualisation après filtrage**» permet de visualiser le résultat de l'algorithme à l'aide du logiciel gephi.

Figure5.5 : Fenêtre de filtrage des communautés chevauchantes

2. Si le fichier d'entré n'est pas correcte l'interface d'erreur suivante va apparaitre

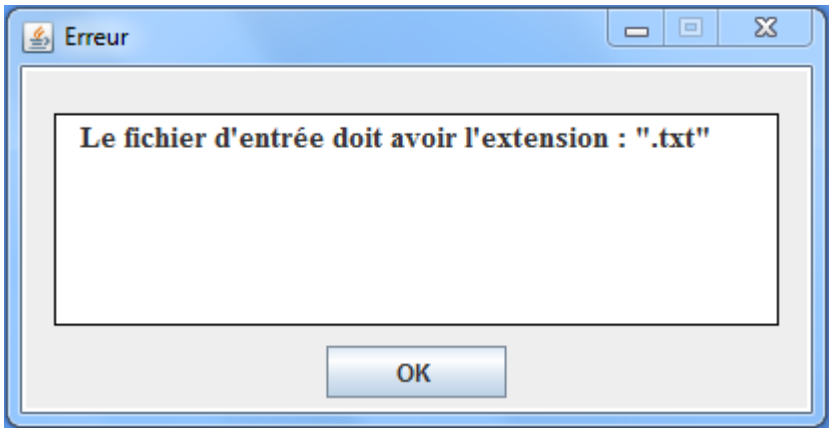


Figure5.6 : Fenêtre d'erreur

4. Résultats expérimentaux

✓ Pour bien comprendre la visualisation des communautés avant et après filtrage nous utiliserons les formats suivants :

La couleur des nœuds

- La couleur d'un nœud est définie par rapport à sa famille (on a deux familles de nœuds rouges et bleus).
- Les nœuds gris sont des nœuds chevauchants.
- Les nœuds de la même communauté ont la même taille.
- Un nœud est représenté par l'identifiant de la communauté à laquelle il appartient et son identifiant séparé par un « / » .
- Après le filtrage un nœud chevauchant est représenté par les identifiants des communautés aux quelles il appartient séparés par des « , » et son identifiant séparé par un « / ».

4.1. Résultats avec un réseau réel

➢ **Le réseau Southern Women**

Southern Women est un réseau social comportant 18 nœuds femmes et 14 nœuds événements, avec 89 arêtes reliant les nœuds femmes et les nœuds événements. Un nœud femme est relié à un nœud événement si la femme a assisté à l'événement correspondant. Ce réseau a été beaucoup étudié par Davis dans le cadre d'une étude approfondie de classe et de race dans le Sud des USA.

D'après la structure de communauté suivante, on voix bien que les communautés 3, 4 , 5, 7, 8 contiennent que de nœuds chevauchants , elles doivent être supprimé selon les étapes de algorithme.

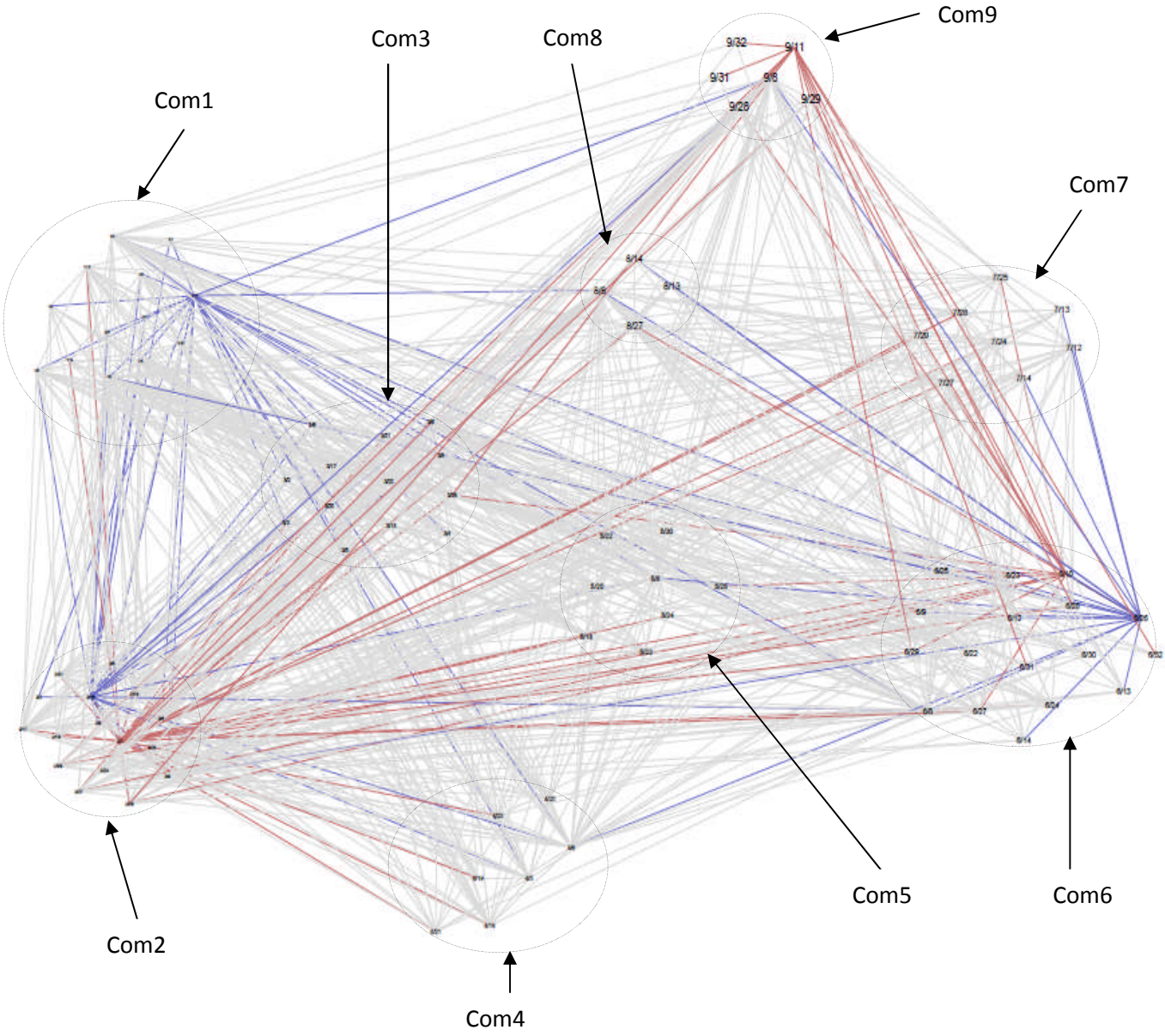


Figure5.7 : Les communautés chevauchantes de la décomposition associée au réseau Southern Women avant filtrage.

Résultat de filtrage de la décomposition Southern Women:

- Nbre_Com_avant filtrage:9
- MQBover_Avant filtrage: 0.12199223
- Nbre_Com_Après filtrage:4
- MQBover_Après filtrage: 0.4874178
- Après filtrage on obtient la communautés 1, 2,3,4.

- La majorités des nœuds chevauchants appartiennent à deux communautés, sauf le nœud chevauchant 8 qui appartient à trois communautés :1, 2, 3.

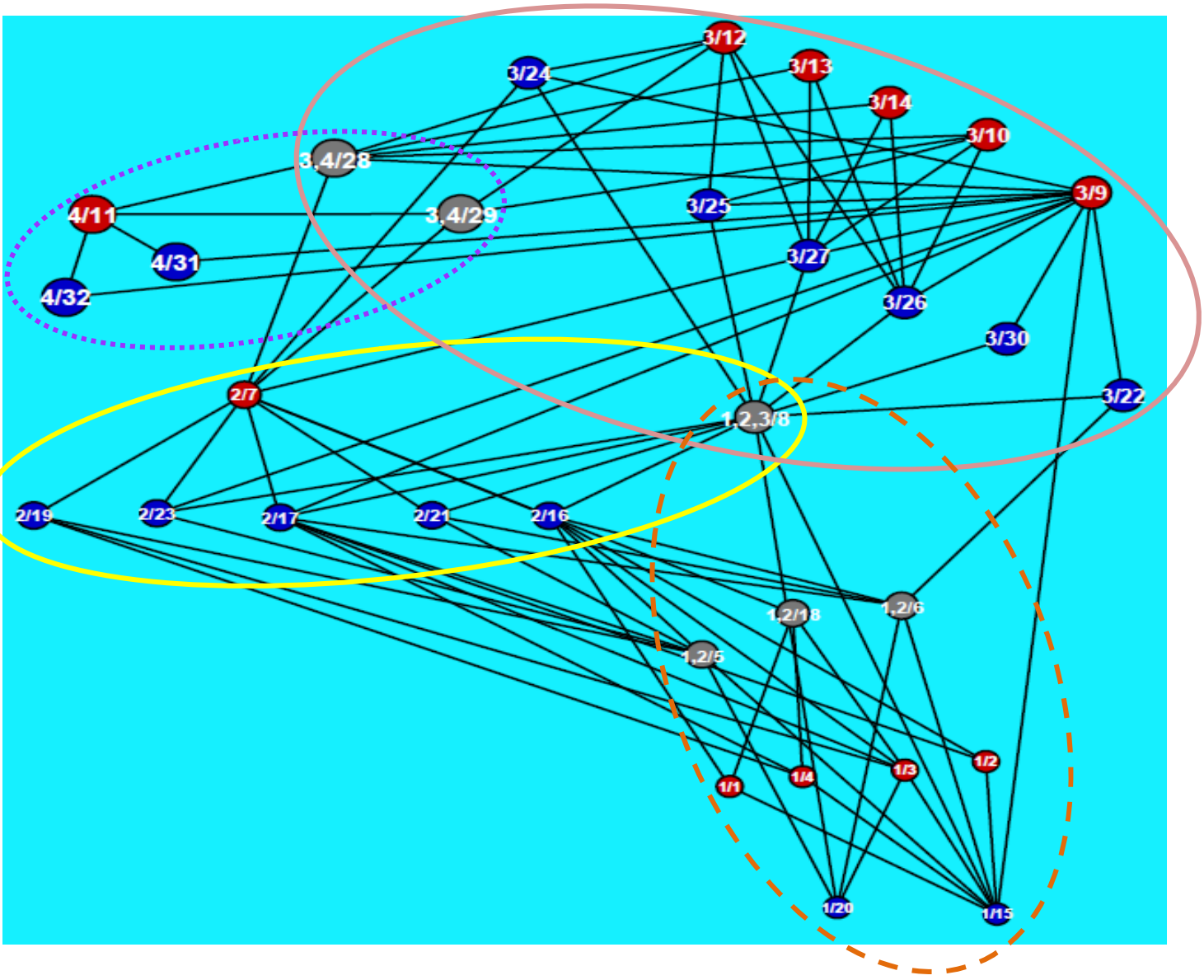


Figure5.8 : Les communautés chevauchantes de la décomposition associée au réseau Southern Women après filtrage

communauté1 :

communauté2 :

communauté3 :

communauté4 :

4.2. Résultats avec un réseau synthétique

Résultat de filtrage d'une décomposition associée au réseau synthétique:

- Nbre_Com_avant filtrage:16
- MQBover_Avant filtrage: 0.3757688
- Nbre_Com_Après filtrage:11
- MQBover_Après filtrage: 0.4326737

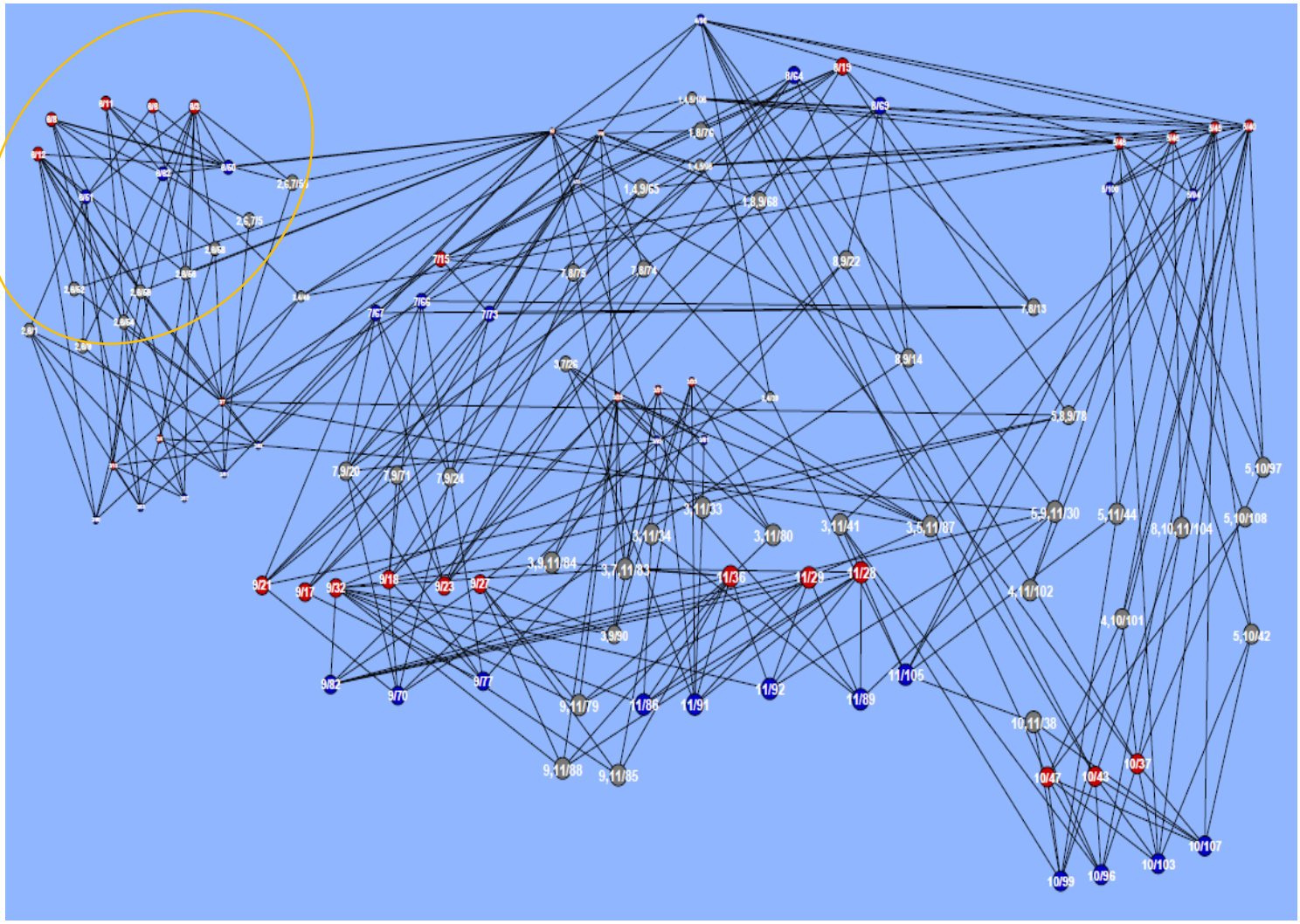


Figure5.9 : Les communautés chevauchantes de la décomposition associée au réseau synthétique après filtrage.

5. Conclusion

On a constaté que l'algorithme de filtrage joue un rôle important, il a amélioré la partition on optimisant la modularité chevauchante.

Sur les réseaux pour lesquels nous avons des décompositions de nœuds en groupes chevauchants présentées dans ce chapitre, les communautés qui n'apportent aucune connaissance sont supprimées, le nombre de nœuds chevauchants est minimisé. Dans tous les cas étudiés la qualité de partition est améliorée par l'algorithme de filtrage. En effet dans chaque cas la modularité avant filtrage est inférieur à la modularité après filtrage.

Conclusion générale

Dans ce projet nous avons implémenté un algorithme de filtrage des nœuds d'une décomposition obtenue à partir de la partition des arêtes (liens) d'un réseau biparti. L'algorithme est destiné aux communautés chevauchantes dans le but d'optimiser la fonction évaluant la décomposition.

Pour estimer l'amélioration de la modularité du réseau, nous mesurons cette dernière avant et après le filtrage. Pour bien voir l'utilité de cet algorithme nous visualisons les communautés avant et après le filtrage.

L'algorithme que nous avons implémenté est très rapide, n'a pas besoin de connaissance à priori de la structure des communautés, comme le nombre de communautés dans la décomposition, les communautés exclusivement chevauchantes, les nœuds chevauchants, la taille des communautés. Nous avons testé cet algorithmes sur deux réseaux, l'un est réel l'autre est synthétique. Dans tous les cas étudiés la qualité de la décomposition a été améliorée.

Dans les communautés chevauchantes l'optimisation de partitionnement pose un problème, ça est dû à la difficulté de travailler et de comprendre la structure de ces communautés. Dans ce but des études sont entrain de se développer pour bien comprendre la notion de communautés chevauchantes et pour améliorer le partitionnement des réseaux.

La réalisation de ce travail m'a permis de bénéficier de plusieurs avantages. Dans un premier temps, j'ai pu approfondir mes connaissances théoriques et pratiques en rapport avec la détection de communautés. Dans un second temps, j'ai pu manipuler certaines techniques de programmation ainsi que le logiciel de visualisation des graphes.

Ce projet informatique reste une modeste contribution dans le domaine de la détection de communautés dans les réseaux. J'espère qu'il sera une source d'inspiration bénéfique pour les futurs étudiants qui abordent ce genre de thème dans leur conduite de projet informatique.

Les références bibliographiques

[1] Burcu Kantarcı , Mise en place d'une Plateforme pour la détection de communautés dans les réseaux complexes , 17 Mayıs 2011

[2] PASCAL PONS, Détection de communautés dans les grands graphes de terrain, Thèse de Doctorat, 2007.

[3] CHARLES-EDMOND BICHOT, la méthode de fusion-fission. application au découpage de l'espace aérien, Thèse de Doctorat, novembre 2007

[4] http://bat710.univ-lyon1.fr/~bserrour/recherche_019.htm

[5] DUONG KHANH CHUONG, Partitionnement de Graphe, master informatique.

[6] GEORGE KARYPIS et VIPIN KUMAR, A fast and high quality multilevel schemefor partitioning irregular graphs. s.l. : SIAM journal of scientific computing, 1998.

[7] PATRICK CIARLET, Calcul réparti : de la pratique à la théorie Le partitionnement , Enseignant-Chercheur.

[8] GAREY M, JOHNSON S, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.

[9] LUC BRUN **,**Complexité Algorithmique.

[10] MATHIEU BARTHELEMY ENS DE LYON ,Graph clustering.

[11] Identification multi-échelle de la structure communautaire de très grands graphes VINCENT BLONDEL, JEAN-LOUP GUILLAUME,RENAUD LAMBIOTTE et ETIENNE LEFEBVRE.

[12] NICOLAS KRÜMMENACKER, Heuristiques de conception de topologies réseaux: application aux réseaux locaux industriels, Thèse de doctorat, 2002.

[13] http://www.metaheuristics.net .

[14] TROUDI FATIHA, Proposition d'un algorithme évolutionnaire multi objectif , thèse de Magister en Informatique, 2006**.**

[15] Walid TFAILI, Conception d'un algorithme de colonie de fourmis pour l'optimisation continue dynamique, Thèse de doctorat, 2007 .

[16] CHRISTOPHE CRESPELLE , Représentations dynamiques de graphes, Thèse de doctorat, 2007 .

[17] ANIS ISMAIL, Communautés dans les réseaux sémantiques pairs-à-pairs, Thèse de doctorat,2010.

[18] JEAN-BAPTISTE ANGELELLI, ALAIN GUENOCHE ET LAURENCE REBOUL, Détection de communautés, disjointes ou chevauchantes, dans les réseaux.

[19] JÖRG REICHARDT and STEFAN BORNHOLDT, Detecting fuzzy community structures in complex networks with a Potts model. Physical Review Letters, 2004.

[20] MAURICIO G. C. RESENDE, Detecting dense subgraphs in massive graphs. In 17th International Symposium on Mathematical Programming, 2007.

[21] EMMANUEL NAVARRO -REMY CAZABET ,Détection de communautés, étude comparative sur graphes réels.

[22] JEFFREY BAUMES, MARK GOLDBERG, and MALIK MAGDON-ISMAIL, Efficient Identification of Overlapping Communities.

[23] Melle. AIT YAKOUB Zina ,Melle. BELHADI Rabia, Détection de communautés dans les réseaux complexes en utilisant les algorithmes génétiques**,** Thèse de Master, 2011.

[24] Projet ISICIL, Analyse des réseaux sociaux et web sémantique,2009

[25] T. COLOMBO. Algorithmes pour la recherche de classes de gènes en relations fonctionnelles par l'analyse de proximités et de similarités de séquences, 2004.

[26] L. DENOEUD, I. CHARON, A. GUÉNOCHE, and O. HUDRY. Overlapping clustering in a graph and application to protein interactions. In *ALIO/EURO conference on Combinatorial Optimization*. Paris, 2005.

[27] NEWMAN, M.E.J. "Fast algorithm for detecting community structure in networks",Physical Review E, 69:066133, 2004.

[28] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen, and E. R. Gansner (1998). Using Automatic Clustering to Produce High-Level System Organizations of Source Code.In IEEE Proc. Int. Workshop on Program Understanding (IWPC'98), pp. 45–53.

[29] R. Bourqui and D. Auber .Analysis of 4-connected Components Decomposition for Graph Visualization.Technical report, LaBRI (http://www.labri.fr/).

[30] SANTO FORTUNATO**,** Community detection in graphs**.**