REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU

FACULTE DU GENIE ELECTRIQUE ET D'INFORMATIQUE

DEPARTEMENT INFORMATIQUE

MEMOIRE

De fin d'études en vue de l'obtention d'un
Master en Informatique
Spécialité: Systèmes informatiques

Thème :

Optimisation locale de la modularité chevauchante d'une décomposition de nœuds d'un réseau biparti

Réalisé et présenté par :

Mlle. OUCHENE Nadia

Mlle. SABI Nawal

Dirigé et proposé par :

Mr. AÏT ELHADJ

Promotion 2011/2012

Remerciements

Nous remercions dieu le tout puissant pour nous avoir donné le courage, la patience et la volonté pour la réalisation de ce modeste travail.

Nos plus vifs remerciements à Madame Aït El hadj pour nous avoir proposé ce thème. Nous lui somme très reconnaissantes pour ses remarques pertinentes, ses conseils judicieux et surtout sa disponibilité à tout moment.

Nous remerciements vont également aux membres du jury, devant qui nous avons l'honneur d'exposer notre travail, et qui ont pris la peine de lire ce mémoire pour juger son contenu.

Nos plus vifs remerciements vont aussi à M^lles : AIT YAKOUB Zina, SEHAKI hayat et MEGHENEZ Zahra pour l'aide précieuse qu'elles nous ont apportées.

Nos remerciements, vont aussi à tous ceux qui ont contribué, du près ou du loin, à la réalisation de ce modeste travail.

Dédicaces

« Louange à dieu, le seul et unique »

- *A les deux âmes, les plus chers au monde : ma mère et mon père, à qui je dois mon existence et mes succés.*

J'espère qu'un jour, je serai capable de leur donner au moins le minimum, car quoique je fasse j'arriverai jamais à leur rendre tout.

- *A mes très chères sœurs : Souad, Naima, Karima et leurs maris.*
- *A mes adorables frères Karim et Abdou*
- *A la personne avec qui je souhaite partager la charge da la vie, mon cher fiancé Yakoub.*
- *A ma future belle famille, en particulier, nana hanifa et yamina.*
- *A mes très chères amies : Hayat, zahra, lila, saliha, samia, qui m'ont beaucoup soutenu, et toutes leurs familles.*
- *A la chère coca et son adorable maman.*
- *A mes tantes et oncles, cousins et cousines, en particulier, ma chère zakia.*
- *A ma chère amie Nadia et toute sa famille.*
- *A tous ceux m'ont soutenu.*

Je dédié ce modeste travail.

Nawal

Dédicaces

« Louange à dieu, le seul et unique »

Je dédie ce modeste travail :

- *À les deux âmes, les plus chers au monde : ma mère et mon père, à qui je dois mon existence et mes succés,*

J'espère qu'un jour, je serai capable de leur donner au moins le minimum, car quoi que je fasse j'arriverai jamais à leur rendre tout.

- *À mon adorable frère Moh areski, pour sa patience et ses encouragements,*
- *À mon frère Hamid pour son soutient,*
- *À mon frère Hacène, seulement pour son engagement indéfectible à mes cotés,*
- *À ma très chère sœur Malika et son mari Samir,*
- *À mon oncle Amar, sa femme Fatiha et leurs deux filles : Yasmine et Nesrine,*
- *À mon oncle Mohamed, sa femme Karima et leurs enfants : Yani, Mayes et Thanina,*
- *À mon adorable Jugurtha,*
- *À ma tante Fatma,*
- *À ma chère amie Nawal et sa famille,*
- *À tous mes amis(es) en particulier mon meilleur ami Mohand et tous les gens qui m'aiment,*

Merci à tous et à toutes.

Nadia

Sommaire

Liste des figures

Liste des tableaux

Résumé

Nous nous situons dans le contexte d'optimisation de la modularité recouvrante d'une décomposition de nœuds en communautés chevauchante, qui est le résultat du partitionnement des arêtes d'un graphe biparti. Le travail qui nous a été proposé dans le cadre de ce mémoire est d'implémenter un algorithme de filtrage et d'optimisation de la modularité de Mancoridis adaptée au graphe biparti.

Mots clés : détection de communautés, détection de communautés chevauchantes, graphe biparti, Optimisation de la modularité chevauchante.

Introduction générale

Dans les grands graphes ou réseaux, la détection de sous-ensembles de sommets plus densément connectés que d'autres, appelés communautés qui jouent un rôle important dans l'organisation ou la structuration de ces réseaux. Dans ces derniers, des nœuds peuvent appartenir à plusieurs communautés, ce qui implique l'apparition de communautés chevauchantes.

La détection de communautés chevauchantes, est une problématique que l'on retrouve dans plusieurs disciplines telles que la biologie (réseaux d'interactions entre protéines), l'informatique (réseaux Web) la recherche opérationnelle et en sociologie (réseaux sociaux)…etc. Des progrès récents dans l'analyse de ces réseaux indiquent que l'identification de ces communautés chevauchantes est importante pour comprendre la structuration et les fonctionnalités de ces réseaux. Une catégorie spéciale de ces réseaux est les réseaux bipartis, dont la détection de telles communautés reste à ce jour peu explorée. En revanche la course à des algorithmes (méthodes) de détection de communautés, que se soient disjointes ou chevauchantes est toujours continue. La plupart de ces méthodes se basent sur l'optimisation des fonctions objectives telle que la modularité qui a été largement utilisée dans les études récentes.

Le travail qui nous a été proposé dans le cadre de ce mémoire est d'implémenter un algorithme d'optimisation de la modularité et de filtrage de nœuds d'une décomposition en communautés chevauchantes, qui est le résultat d'un partitionnement d'arêtes d'un graphe biparti, pour cela, notre mémoire est organisé en cinq chapitres. Dans le premier chapitre, on présente des notions générales sur les graphes, puis on introduit la problématique de détection

de communautés et les différentes méthodes utilisées. Le deuxième chapitre présente la problématique de détection de communautés chevauchante et quelques méthodes utilisées. Le troisième chapitre aborde une mesure de qualité pour l'évaluation de découpage du graphe en communautés, qui est la modularité de Mancoridis et son adaptation aux graphes bipartis, ensuite cite quelques méthodes pour son optimisation. Le quatrième chapitre décrit la structuration des données et les différents algorithmes implémentés pour la réalisation de notre application. Le dernier chapitre est consacré à la description de mise en œuvre de notre logiciel. Enfin nous terminons par une conclusion générale.

Chapitre I

Détection de communautés

Introduction

La détection de communautés est une problématique assez ancienne, qui tire ses origines du problème de partitionnement de graphe en théorie des graphes. Elle consiste à découper un graphe en différentes parties disjointes qui satisfont certaines contraintes.

Nous nous intéressons dans ce premier chapitre à définir quelques notions sur les graphes, puis à présenter la problématique de détection de communautés dans les réseaux, en citant des domaines concernés et des méthodes utilisées.

I. Généralités, définitions et notations sur les graphes

Définition 1 (Graphe)

Un graphe simple G est défini par le couple (V, E) tel que V est l'ensemble de points appelés sommets ou nœuds, mis en relation par l'ensemble E de liens appelés arcs ou arêtes du graphe, suivant qu'ils sont orientés ou non.

Définition 2 (Graphe k_parti)

Un graphe est dit k-parti si l'ensemble de ses sommets est décomposable en k parties telles que toute paire de sommets d'une même partie ne soit pas reliées par une arête.

Définition 3 (Graphe biparti)

Un graphe biparti est une catégorie spéciale de graphe k_parti, 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.

Autrement dit : un graphe G=(S, E) est biparti si et seulement s'il existe deux sous-ensembles U et V tel que U ∪ V =S, qui vérifié : ∀ (u, v) ∈ ExE , u∈U ⇔ v∈V.

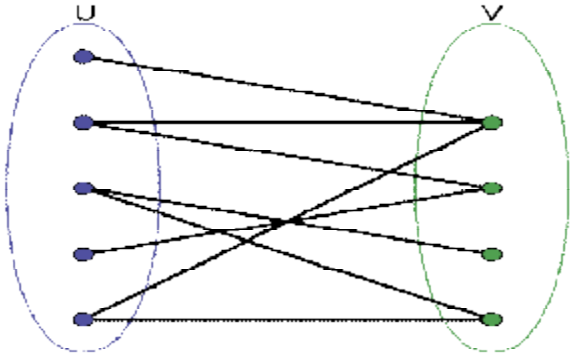


Figure I.1. Exemple de graphe biparti.

Définition 4 (Graphe valué ou pondéré)

On dit qu'un graphe G = (V, E) est valué (ou pondéré) si a chaque élément a de V est associé une valeur entière appelée poids(a).

Définition 5 (arête incidente à un sommet)

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

Définition 6 (Degré d'un sommet)

Le degré d'un sommet i est deg(i), il correspond au nombre d'arêtes qui lui sont incidentes.
Un sommet de degré 0 est dit sommet isolé.

Définition 7 (Adjacence)

Soient un graphe G = (V, E) et une arête u = (v, v′) ∈ ExE. On dit que les sommets v et v′ sont des sommets adjacents.

Définition 8 (Matrice d'adjacence et des degrés)

Soit un graphe simple G = (V, E).

La matrice d'adjacence M_{Adj} est de dimensions $n_v \times n_v$, telle que $\forall (i, j) \in \{1,.., n_v\}^2$:

$(M_{Adj})_{i\,j}$ =

La matrice des degrés MDeg de dimensions n_v x n_v, telle que $\forall (i, j) \in \{1,.., n_v\}^2$:

$(M_{Deg})_{ij=}$

Définition 9 (Matrice Laplacienne)

Soit un graphe G = (V, E). La matrice laplacienne de g est définie par :

$M_{Lap} = M_{Deg} - M_{Adj}$

Définition 10 (densité)

La densité d'un graphe G= (V, E) est le rapport |E |/ | V | [1], tel que | E | est le nombre des arêtes de G et | V | est le nombre de ses sommets.

-Un graphe peu dense contient peu d'arêtes (arcs)

-Un graphe dense contient beaucoup d'arêtes (arcs)

Définition 11 (Graphe de lien)

Soit G un graphe simple, on appelle graphe-arête (ou graphe représentatif des arêtes) de G, le graphe G' obtenu de la façon suivante :

- les sommets de G' représentent les arêtes de G.

- deux sommets de G' sont reliés si et seulement si les arêtes correspondantes de G sont adjacentes.

II. Détection de communautés dans les réseaux

1. Définition

La détection de communautés, consiste à scinder un graphe G (ou réseau) en k partitions (sous-réseaux) disjoints, sans connaître à priori ni la taille ni le nombre k de communautés (partitions). D'un point de vue mathématique, on peut partitionner les sommets ou bien les arêtes. Par contre, dans la plupart des applications, on ne s'intéresse qu'au partitionnement des sommets du graphe.

Soient un graphe G = (V, E) et un ensemble de k sous-ensembles de V, noté $P_k = \{V_1, V_2,..,V_k\}$. On dit que P_k est une partition de G si :

1. Aucun élément de P n'est vide

   $\forall\ i \in \{1,\ldots,k\}$ **à** $V_i \neq \emptyset$

2. L'union des éléments de P est égale à V

   $\bigcup_{i=1}^{k} V_i = V$

[1] La notation | | se lit cardinal, est égale au nombre total des éléments d'un ensemble.

3. Les éléments de P_k sont deux à deux disjoints.

   $\forall\, i, j \in \{1,\ldots,k\}^2$, $i \neq j$ alors $V_i \cap V_j = \emptyset$

La notion de communauté dans un graphe est cependant difficile à définir formellement, il n'existe pas à ce jour de définition satisfaisante.

C'est pourquoi toutes les approches récentes ont utilisé une approche moins formelle et plus intuitive de la notion de communauté [3, 4].

Une communauté est alors vue comme un ensemble de sommets dont la densité de connexions internes est plus forte que la densité de connexions vers l'extérieur.

Une de ces approches consiste à quantifier la notion intuitive de communautés, et cela en utilisant des fonctions de qualité (dites aussi fonctions objectives), qui permettent de mesurer la qualité d'une partition en communautés. 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 de qualité tiennent généralement compte de rendre la densité plus forte à l'intérieur de chaque communauté (intra communauté) et plus faible à l'extérieur (inter communauté), la fonction la plus communément utilisée étant la modularité, elle sera présentée dans le chapitre 3. [4, 7]

Notons enfin que cette définition impose que les communautés soient disjointes, or en, pratique rien n'empêche l'existence de communautés qui se chevauchent. La détection de telles communautés est un problème bien plus difficile que la détection de communautés disjointes. Ce problème a été très peu traité et sera l'objectif du chapitre suivant.

2. Domaines d'Applications

La détection de communautés devient de plus en plus une question importante et vise de nombreux domaines d'applications tels que:

- **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.

- **La conception de circuits électroniques intégrés**

Le problème de conception de circuits électroniques intégrés est de diviser les composants aux sous-circuits tels que le nombre de connexion entre ceux-ci soit minimal. C'est une tache de plus en plus complexe car les composants sont de plus en plus petits et le nombre de composants augmente 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 repartir 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 correspondantes aux communications nécessaires à la programmation parallèles

- **La segmentation d'image**

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

3. Complexité

La complexité d'un algorithme permet de quantifier mathématiquement le niveau de difficulté lors de la résolution de problèmes, selon des critères de temps d'exécution, de nombre d'itérations (ou d'opérations arithmétiques) ou encore d'espace mémoire nécessaire.

Ces grandeurs sont indiquées en fonction de la taille des données du problème à résoudre. La théorie de la complexité établit une classification des problèmes en fonction des algorithmes susceptibles de les résoudre, les machines de Turing constituent un étalon fréquemment utilisé. Sans entrer dans les détails, nous distinguons deux grandes classes P et NP des problèmes. Une description plus complète des différentes classes des problèmes d'optimisation combinatoire peut être trouvée dans. [3, 6]

Un problème est dit de classe P s'il peut être résolu en un temps polynomial par un algorithme déterministe dont le temps d'exécution est de complexité polynomiale (P signifiant alors *Polynomial time).* Les problèmes de classe P sont considérés comme des problèmes faciles. C'est le cas notamment du problème de la connectivité d'un graphe.

La classe NP est une classe plus riche de problèmes pour lesquels on peut construire un algorithme non déterministe dont le temps d'exécution est de complexité polynomiale *(NP* signifiant *Nondeterministic Polynomial time).*

Un problème *NP* est dit *NP-complet* si tout problème de la classe NP peut lui être réduit : c'est le cas de nombreux problèmes considérés comme intraitables en pratique (pour lesquels les algorithmes déterministes connus sont exponentiels).

Le problème de partitionnement est un problème NP-complet [4, 6], il peut être résolu par :

- **Méthode exacte** : Dans le pire des cas, un tel algorithme aurait besoin d'énumérer toutes les solutions candidates de l'espace de recherche. Par conséquent, seule une très petite instance du problème peut être résolue à l'intérieur d'un temps raisonnable et les grandes instances sont impraticables.

- **Les métaheuristiques** : Elles ne garantissent pas de trouver de manière exacte l'optimal mais qui ont une bonne chance de trouver une bonne solution qui est proche de l'optimal en un temps raisonnable. Ces méthodes approchées constituent une alternative très intéressante pour traiter les problèmes NP-complet.

III. Les méthodes de détection de communautés

Les méthodes de détection de communautés ont fait l'objet de nombreux travaux, de ce fait, leur classement a été pris en différents points de vue.

La plupart de méthodes de détection de communautés consistent à déterminer une partition des sommets du graphe optimisant un certain critère de qualité d'un partitionnement, défini à partir de la structure du graphe. [2]

Dans [1] les méthodes sont regroupées selon : partitionnement contraint (principale caractéristique est de rechercher des partitions de tailles égales) ou non contraint (la principale caractéristique est de rechercher des partitions de tailles différentes).

Dans [6], les méthodes sont regroupées comme suit :

- les méthodes **"ascendantes"**: partant de la partition atomique, on réunit deux classes à chaque itération. Les classes à fusionner sont celles qui promettent une modularité maximum
- les méthodes **"descendantes"** : dans lesquelles on part du graphe entier. A chaque itération, on scinde une classe en deux sous-classes disjointes suivant des principes similaires.

Les méthodes précédemment citées 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.

Dans [5] les méthodes sont regroupées dans deux classes : globales ou locales.

- Méthodes **"globales"**: reposent sur l'exploitation de la globalité de l'information disponible sur le graphe en tentant de regrouper les sommets en prennent en compte leur position géographique.
- Méthodes **"locales"**: reposent sur l'exploitation de déplacement de proche en proche dans le graphe en tentant de regrouper les sommets fortement connectes sans prendre en compte leur position géographique).

Dans [2], les méthodes sont regroupées en classes d'approches classiques (qui regroupent Les premiers algorithmes développés pour les problèmes de partitionnement de graphe), et en classe d'approches récentes qui regroupent les algorithmes récents (qui sont spécialement développés pour la détection de communautés).

On s'intéresse dans notre travail, à cette dernière classification des méthodes, en citant quelques algorithmes destinés au partitionnement de graphes, mais se basant beaucoup plus sur ceux de détection de communautés.

1. Approches classiques

1.1. Algorithme de colonies de fourmis

L'algorithme de colonies de fourmis simule la recherche de nourriture des fourmis. Cette idée a été présentée première fois par A.Colorni, M.Dorigo et V.Maniezzo [1]. Une fourmi pose de la phéromone quand elle se déplace. Les fourmis se dirigent de manière probabiliste en comptant de la quantité de phéromone qui est autour d'elles. Plus la quantité de phéromone indiquant un chemin est grand, plus des fourmis ont tendance à suivre ce chemin. Comme la phéromone s'évapore progressivement, le choix probabiliste que prend une fourmi pour choisir son chemin évolue continuellement. Grâce à l'utilisation de phéromone, les fourmis peuvent collectivement trouver le plus court chemin entre deux points. L'algorithme de colonies de fourmis est bien utilisé à nombre de problèmes d'optimisation et il a été utilisé pour le problème de partitionnement de graphe dans les dernières années.

1.2. Greedy Graph Growing Algorithm (GGGP)

Le *Greedy Graph Growing Algorithm* (GGGP) a été introduit dans [1]. Cet algorithme a pour but de 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. Autrement dit, c'est le sommet qui a le gain de la fonction objectif 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.

Procédure GGGP (G= (V, E)) :

```
Debut
Prendre au hasard un sommet s0 Î R
  P ß {V0}
    F ß {V  Î R tel que (v, v0) Î E }
    Calculer les gains de F
    Tant que poids(P) < ½ poids(V) faire
          Prendre le sommet Vi Î F de gain maximal
          Déplacer Vi de F vers P
          Pour toute arête (v, vi) dans E faire
                si  v Î F alors
                      mettre à jour le gain de V
                sinon si V Î E alors
                      ajouter V à F
                      calculer le gain de V
                fin si
          fin pour
    fin tant que
   retourner P
```

1.3. Méthode spectrale

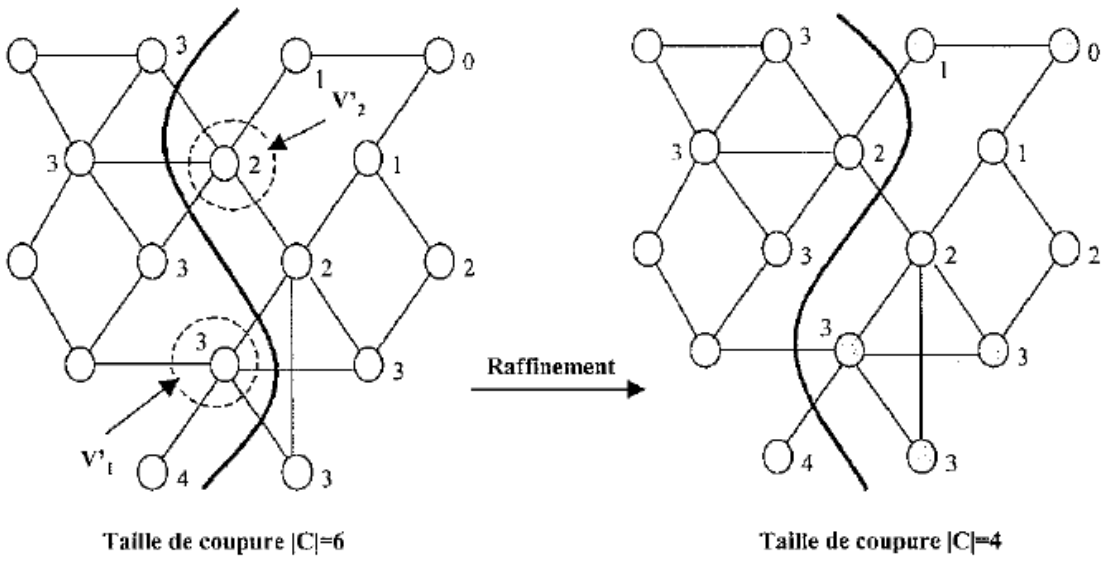
L'utilisation de la méthode spectrale pour résoudre les problèmes de partitionnement de graphe est très ancienne. Les premiers articles sont proposés depuis les années 70 par W.Donath et A.Hoffman [2]. Cette méthode est basée sur le théorème spectral de l'algèbre linéaire qui permet d'affirmer la diagonalisation 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 = λx. Résoudre ce système numérique consiste à trouver une base orthogonale de vecteurs propres de la matrice M.

1.4. Algorithme de Kernighan-Lin

Kernighan et Lin ont proposé en 1970 une des premières méthodes de partitionnement de graphes [6]. Spécialement conçue pour le problème de bipartition de graphes, cette méthode utilise une partition initiale, et tente de l'améliorer tout en conservant le même nombre de sommets dans chacun des sous-graphes. Cette technique est souvent employée lors de phases dites de *raffinement* dans d'autres algorithmes de recherche. L'idée est la suivante:

Soit une bissection $\{V_1, V_2\}$ d'un graphe G = {V, E}, le traitement par l'algorithme de K-L consiste à trouver deux sous-ensembles V'_1 et V'_2 (avec $|V'_1| = |V'_2|$) contenus respectivement dans V_1 et V_2 puis de les permuter entre les partitions de façon à réduire au maximum la taille de la coupe (Figure 1.1). Ces permutations se font jusqu'à ce que l'on ne puisse plus améliorer la qualité de la partition. L'identification des ensembles V'_1 et V'_2 est basée sur un algorithme glouton recherchant de manière itérative une paire de sommets (chacun appartenant à des sous-graphes différents), puis à les permuter si cela diminue la taille de la coupe. La première paire sélectionnée est celle qui apporte la plus forte amélioration et ainsi de suite, sachant qu'un sommet ne peut être déplacé qu'une seule fois durant une même procédure. Lorsque toutes les paires ont été trouvées, une nouvelle procédure peut alors commencer.



FigureI.2. Algorithme de Kernighan-Lin

2. Approches récentes

2.1. Approches séparatives

L'idée commune de toutes ces méthodes est d'essayer de scinder le graphe en plusieurs communautés en retirant progressivement les arêtes reliant deux communautés distinctes. Les arêtes sont retirées une à une, à chaque étape les composantes connexes du graphe obtenu sont identifiées à des communautés. Le processus est répété jusqu'au retrait de toutes les arêtes. On obtient alors une structure hiérarchique de communautés. Les méthodes existantes diffèrent par la façon de choisir les arêtes à retirer. [2]

2.1.1. Algorithme de Givran et Newman

Cette approche retire les arêtes de plus forte centralité d'intermédiarité. Cette centralité est définie pour une arête comme le nombre de plus courts chemins passant par cette arête. Il existe en effet peu d'arêtes reliant les différentes communautés et les plus courts chemins entre deux sommets de deux communautés différentes ont de grandes chances de passer par ces arêtes. Un algorithme calculant la centralité de toutes les arêtes en O(mn) est proposé. Ce calcul est effectué à chaque étape sur le graphe obtenu après retrait des arêtes. La complexité de l'algorithme est donc $O(m^2n)$. Une variante considérant des marches aléatoires à la place des plus courts chemins est aussi introduite. Elle donne des résultats légèrement meilleurs mais demande encore plus de calculs. [6]

2.1.2. Algorithme de Radicchi et al

La détection des arêtes intercommunautaires ici est basée sur le fait que de telles arêtes sont dans des zones peu regroupée. Un coefficient de regroupement (d'ordre g) d'arêtes est défini comme étant le nombre de cycles de longueur g passant par l'arête divisé par le nombre total de tels cycles possibles (étant donné les degrés des sommets). Cet algorithme retire donc à chaque étape l'arête de plus faible regroupement (d'un ordre donné). Chaque suppression d'arête ne demande alors qu'une mise à jour locale des coefficients de regroupement, ce qui lui permet d'être bien plus rapide que le précédent algorithme. [2]

2.1.3. Algorithme Fortunato et al

Il s'agit d'une variante de l'algorithme de Girvan et Newman basé sur une autre notion de centralité : la centralité d'information. Les auteurs définissent l'efficacité de communication E_{ij} entre deux sommets i et j du graphe comme étant l'inverse leur distance. L'efficacité E du réseau est alors définie comme la moyenne des E_{ij} pour tous les couples de

sommets. Enfin, la centralité d'information d'une arête est définie comme étant la diminution relative de l'efficacité du réseau lorsque l'on retire cette arête du graphe. Cette approche donne de meilleurs résultats mais a une très mauvaise complexité en $O(m^3n)$. [2][6]

2.2. Approches agglomératives

L'idée commune de ces méthodes est d'utiliser une approche s'apparentant à celle du clustering hiérarchique utilisée dans le partitionnement des graphes, dans lesquelles les sommets sont regroupés itérativement en communautés.

2.2.1. Algorithme de Newman d'optimisation de la modularité

Newman introduit une valeur Q quantifiant la qualité d'une partition du graphe en communautés, qui est la modularité. L'algorithme fusionne alors à chaque étape les communautés permettant d'avoir la plus grande augmentation de la modularité.

Pour améliorer les performances de l'algorithme, seules les communautés ayant une arête entre elles peuvent être fusionnées à chaque étape. Chaque fusion se fait en O(n) et la mise à jour des valeurs des variations de Q (pour chaque nouvelle fusion possible) peut être effectuée facilement en O(m). La complexité globale est alors O(mn). Cette méthode est rapide mais donne moins de bons résultats que les autres. [2] La modularité utilisée dans cette méthode sera détaillée dans le chapitre 3.

2.2.2. Algorithme de Donetti et Munoz

Les auteurs observent que les vecteurs propres de la matrice Laplacienne donnent de l'information sur les communautés. En effet les coordonnés i et j des vecteurs propres correspondant aux plus petites valeurs propres sont corrélées lorsque les sommets i et j sont dans la même communauté. Une distance entre sommets est alors calculée à partir de ces vecteurs propres. Le nombre de vecteurs propres à considérer est a priori inconnu, plusieurs calculs sont successivement effectués en prenant en compte différents nombres de vecteurs propres, et le meilleur résultat est retenu. [5][6]

Conclusion

On a présenté dans ce premier chapitre, des notions de base sur les graphes, puis nous avons introduit par la suite la problématique de détection de communautés. Nous avons présenté aussi un état de l'art des méthodes les plus utilisées dans la résolution du problème de partitionnement de graphes et de détection de communautés.

Notons que la détection de communautés, nous induit à des communautés disjointes, or en pratique, rien n'empêche l'existence de communautés chevauchantes. La détection de ces dernières est l'objectif du chapitre suivant.

Chapitre II

Détection de communautés chevauchantes

Introduction

Les méthodes de détection de communautés citées précédemment, ont l'avantage d'être efficaces et de s'appliquer à des grands graphes, mais elles produisent des partitions en communautés disjointes, ce qui n'est pas toujours désiré ni justifié. De nombreux groupes peuvent se chevaucher comme c'est le cas, par exemple de chercheurs pouvant appartenir à plus d'une communauté de recherche, et leur affectation à un groupe unique n'est pas justifiable. C'est pourquoi le problème de construire des partitions en communautés chevauchantes se pose.

L'objectif principal de ce chapitre est d'introduire la problématique de détection de communautés chevauchantes dans les réseaux, plus spécialement dans les réseaux bipartis, en citant certains domaines concernés et quelques méthodes utilisées.

I. Objectifs et applications de chevauchement de communautés dans les réseaux

Dans les grands réseaux, des nœuds peuvent appartenir à plusieurs communautés, ce qui implique l'apparition de communautés chevauchantes. L'identification de ces dernières est très nécessaire pour comprendre la structure et la fonctionnalité de réseau.

La détection de communautés chevauchantes, est une problématique que l'on retrouve dans plusieurs disciplines telles que la biologie (réseaux d'interactions entre protéines),

l'informatique (le réseau Web consiste en un ensemble de pages connectées par des hyperliens), la recherche opérationnelle, et de même en sociologie (réseaux sociaux).

- ***En Sociologie* :** des individus sont regroupés en communautés liées par des relations sociales, telles que l'amitié, le travail…etc. Comme un individu peut avoir plus d'une relation en même temps, il est affecté à plusieurs groupes d'où le chevauchement des communautés.
- ***En Biologie* :** dans les réseaux d'interactions protéine-protéine, ces dernières ont plusieurs fonctionnalités et peuvent participer dans plusieurs cellules. Dans ce cas il est raisonnable de construire des partitions en communautés chevauchantes.
- ***Dans les réseaux mobiles :*** Le concept de communautés chevauchantes est montré aussi dans ces réseaux qui sont très dynamiques, de ce fait, certains dispositifs actifs peuvent participer à plusieurs groupes en même temps. [12][16]

II. Détection de communautés chevauchantes

1. Définitions

1.1. Détection de communautés disjointes

La détection de communautés disjointes permet de trouver les différentes communautés d'un graphe de sorte que chaque nœud appartient à une et une seule communauté.

Soit un graphe G = (V, E) et un ensemble de k sous-ensembles de V, noté Pk = {V_1, V_2,…, V_k}. On dit que Pk est une partition de G si :

1. Aucun élément de P n'est vide

   $\forall\ i \in \{1,\ldots,k\}$ **à** $V_i \neq \emptyset$
2. L'union des éléments de P est égale à V

   $\bigcup_{i=1}^{k} Vi = V$
3. Les éléments de P sont deux à deux disjoints.

   $\forall\ i, \in \{1,\ldots,k\}, \neq j$ **à** $Vi \cap Vj = \emptyset$

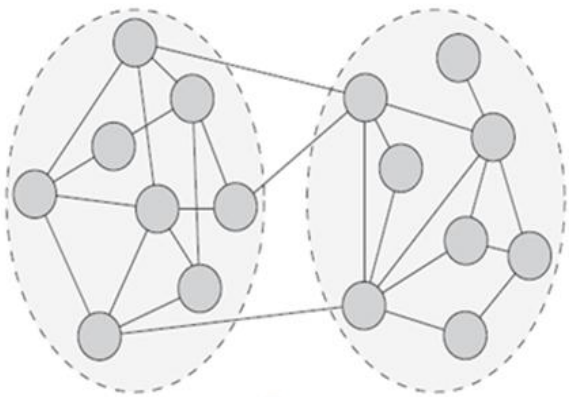


Figure II.1. Exemple de deux communautés disjointes.

1.2. Détection de Communautés chevauchantes

Un système de chevauchement de communautés est un ensemble de communautés interconnectées entre elles. Un nœud peut appartenir à une ou plusieurs communautés.

Soit un graphe G = (V, E) et un ensemble de k sous-ensembles de V, noté Pk = {V_1, V_2,…,V_k}. On dit que Pk est une décomposition de nœuds en communauté chevauchante si :

1. Aucun élément de P n'est vide
   $\forall\ i \in \{1,\ldots,k\}$ **à** $V_i \neq \emptyset$
2. L'union des éléments de P est égale à V
   $\bigcup_{i=1}^{k} Vi = V$
3. Les éléments de P peuvent appartenir à une ou plusieurs communautés.
   $\forall\ i, \in \{1,\ldots,k\}, \neq j$ **à** $Vi \cap Vj \neq \emptyset$

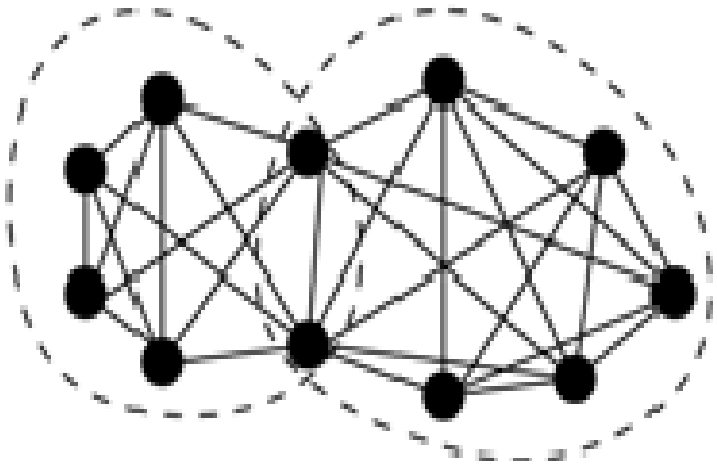


Figure II.1. Exemple de deux communautés chevauchantes.

Des nombreuses méthodes ont été conçues pour détecter la structure des communautés disjointes dans les réseaux, la plupart d'entre elles, sont adaptées pour la détection de communautés chevauchantes. Cependant, peu sont les méthodes destinées à la détection de communautés chevauchantes dans les réseaux bipartis, cette problématique reste récente et en évolution.

III. Méthodes pour la détection de communautés chevauchantes

Ces dernières années ont vu le développement de nombreuses méthodes, qui permettent d'identifier la structure des communautés dans les réseaux. La grande majorité de ces méthodes est destinée à la détection de communautés disjointes. Certaines ont été étendues pour détecter des communautés chevauchantes.

Dans cette section, nous présentons quelques méthodes utilisées dans la détection de communautés chevauchantes dans les réseaux monopartis, suivi d'une méthode dédiée aux graphes bipartis.

1. Détection de communautés chevauchantes dans les réseaux monopartis

1.1. L'algorithme LA

Cet algorithme se base sur l'efficacité de distribution des nœuds, ces derniers sont insérés dans les communautés suivant un certain critère, tel que, un nœud est ajouté à toute communauté si son ajout améliore sa densité, si c'est ne pas le cas une nouvelle communauté est crée pour recevoir ce nœud, dans ce cas chaque nœud appartient à au moins une communauté. [18]

Le temps d'exécution peut être lié en termes de nombre de communautés C, le déroulement de cet algorithme est le suivant :

Algorithme LA :

procédure *LA(G = (V,E),W)*

Debut
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 toute communauté*
if *W (Dj ∪ vi) > W(Dj)* ***then***
Dj ← Dj ∪ vi ;
added ← true ;
if *added = false* ***then***
C ← C ∪{{vi}}; //crée une nouvelle communauté
return C;

Fin.

1.2. OCA (Overlaping Communauties Algorithm)

OCA est un algorithme de détection de communautés chevauchantes, il est basé sur une nouvelle fonction d'optimisation (fitness) pour l'évaluation de la la qualité d'une communauté. Il a été conçu pour distinguer chaque communauté à part de l'ensemble de communautés dispersées au hasard dans un graphe. En particulier, il commence par un ensemble aléatoire de nœuds puis il ajoute (ou supprime) un nœud de l'ensemble restant dont l'addition ou l'élimination d'un nœud implique une augmentation de la fonction d'optimisation. Quand on ne peut ni ajouter ni supprimer n'import lequel nœud, dans ce cas nous avons trouvé un maximum de nœuds local donc une communauté. Cette procédure est répétée jusqu'à ce qu'un critère d'arrêt soit rencontré. [15]

1.3. Algorithme Peacock

Peacock se base sur la transformation du réseau original en un autre réseau plus grand, ce dernier est traité par des algorithmes de détection de communautés disjointes, puis transformer le résultat pour détecter des communautés chevauchantes.

Le principe de cet algorithme est le suivant :

Le réseau initial est transformé en un nouveau réseau plus grand. Chaque étape de la transformation divise un sommet en deux sommets, en supposant que les sommets de réseau original ont été reliés et connectés à celle transformée. Les noms des sommets impliqués dans chaque étape de division seront stockés pour une utilisation ultérieure, par exemple, si le sommet v est divisé en {v, v '} tel que v' est enregistré dans un fichier en tant qu'une copie du sommet v.

En outre, tous les sommets dans le fichier transformé sont renommés en entier, pour la compatibilité avec certains algorithmes de détection de communautés qui imposent cette restriction.

Le réseau transformé est appliqué à un algorithme de détection de communautés, qui produit un ensemble de communautés disjointes. Ces dernières sont converties à un regroupement de communautés chevauchantes par le remplacement des noms de sommets stockés dans le fichier par celles utilisées dans le réseau original. Par exemple, si Peacock divise un sommet v en{v, v '} qui figurent dans les deux communautés disjointes, par le remplacement de v ' par v, le regroupement final comprend v dans les deux communautés, ce qui va donner un chevauchement de communautés.[13]

Les étapes de cet algorithme sont illustrées dans la figure suivante:

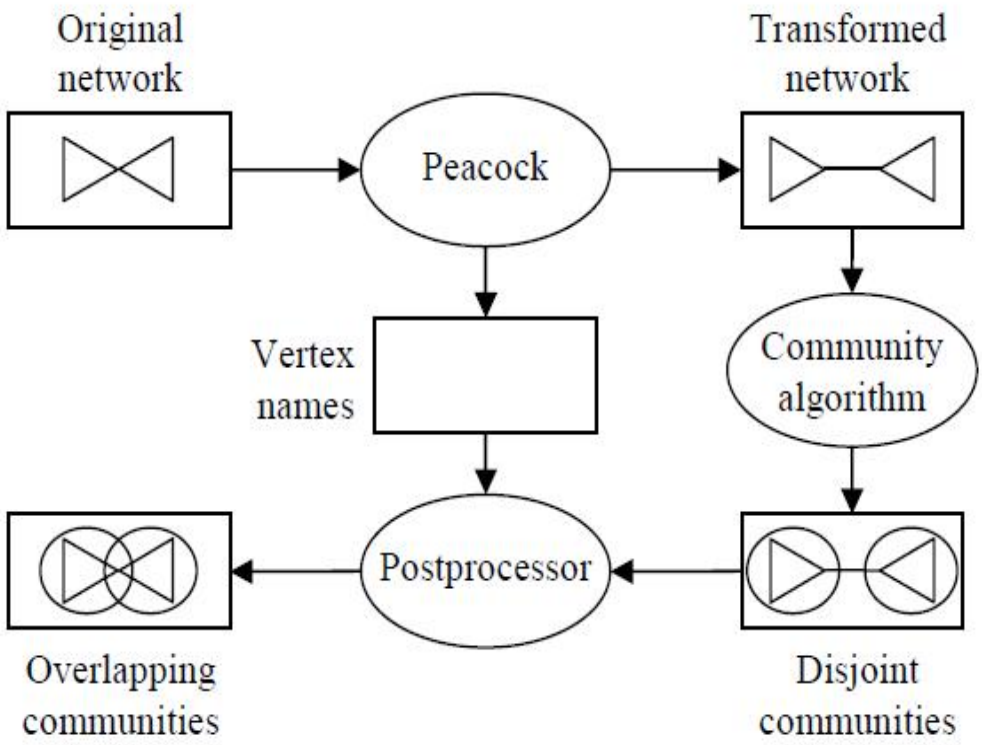


Figure II.3. Système de l'algorithme peacock.

Les méthodes qu'on a citées ci-déçus détectent des communautés chevauchantes dans les graphes monopartis. Il ya aussi des méthodes spéciales pour la détection des communautés chevauchantes dans les réseaux bipartis, ces méthodes se basent sur le graphe de lien et l'algorithme évolutionnaire[2].

2. Détection de communautés chevauchantes dans des réseaux bipartis

Un graphe (ou réseau) biparti est une catégorie spéciale de graphe, où les nœuds peuvent être divisés en deux sous-ensembles (familles) disjoints, tels qu'il ne peut y avoir aucun lien qui relie deux nœuds de la même la famille. Les réseaux auteur-publication, les réseaux acteur-film, les réseaux produit-consommation…etc., sont des exemples concrets de réseaux bipartis. [14]

[2] Les algorithmes évolutionnaire sont une famille d'algorithmes s'inspirant de la théorie de l'évolution pour résoudre des problèmes divers. Ils font évoluer plusieurs solutions pour un problème donné dans le but de trouver une solution optimale, ils s'agirent donc de résoudre des problèmes d'optimisation. [3]

Récemment, certains auteurs ont étudié la problématique de détection de communautés dans les réseaux bipartis. Cependant, la notion de communauté dans de tels réseaux n'a pas été abordée de la même manière.

En effet, une communauté d'un réseau biparti est composée de nœuds de même type. Par exemple dans un réseau auteur-publication, les auteurs forment les communautés auteur et les publications forment les communautés publication. Par conséquent, des algorithmes consistent à détecter des communautés de nœuds d'un même type à la fois.

En revanche, une autre approche considère une communauté de réseau biparti comme un groupe de nœuds densément connectés, qui est le point de vue traditionnel dans un réseau simple [3]. Ce point de vue est celui qu'adopte la méthode que nous allons implémenter.

De nombreux chercheurs scientifiques s'intéressent à ce type de réseaux et ils se concentrent sur l'exploitation directe de chevauchement de communautés dans les réseaux bipartis.

Dans ce contexte, un algorithme dit génétique[3], est adapté pour la détection de communautés chevauchantes dans les réseaux bipartis. Les principales étapes de l'algorithme sont abordées dans la section suivante, et son principe est détaillé dans [3].

2.1. Algorithme génétique pour la détection de communautés chevauchantes à partir d'un graphe de lien

Le principe général de cet algorithme 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 génétique proposé par « MurselTasgin » et « HalukBingol » pour partitionner le graphe de lien obtenu en sous ensembles. Et enfin, en appliquant un algorithme d'optimisation locale appliqué sur la partition du graphe d'arête, et il obtient des communautés recouvrantes.

a) Construction du graphe de lien

Construire le graphe de lien G' dont les sommets sont les numéros des arêtes du graphe initial G et dont les arêtes sont les liens d'incidence entre les arêtes de G.

[3] Les algorithme génétique sont des cas particulier des algorithmes évolutionnaires et sont les plus populaires de ces algorithmes.

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

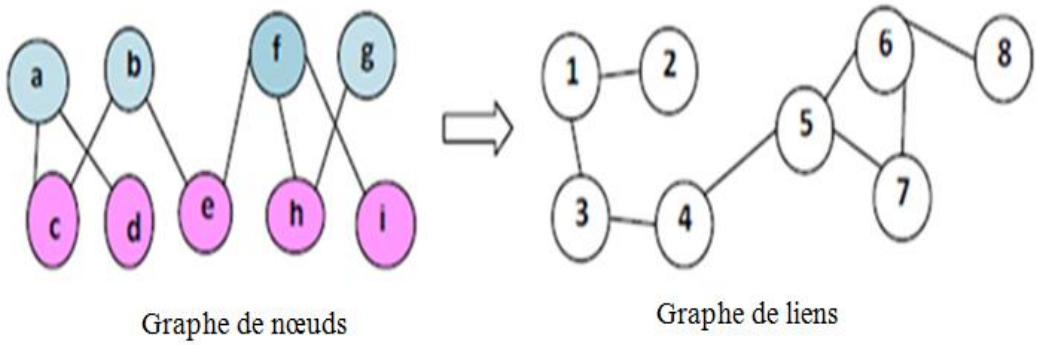


Figure II.4. Transformation d'un graphe de nœuds au graphe de lien.

b) Partitionnement du graphe de lien

Cette étape consiste à partitionner G' (ensemble des arêtes de G) en p groupes disjoints, en utilisant l'algorithme génétique.

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 figure ci-dessous montre un exemple de partitionnement de graphe de lien présenté dans la figure précédente en deux communautés disjointes.

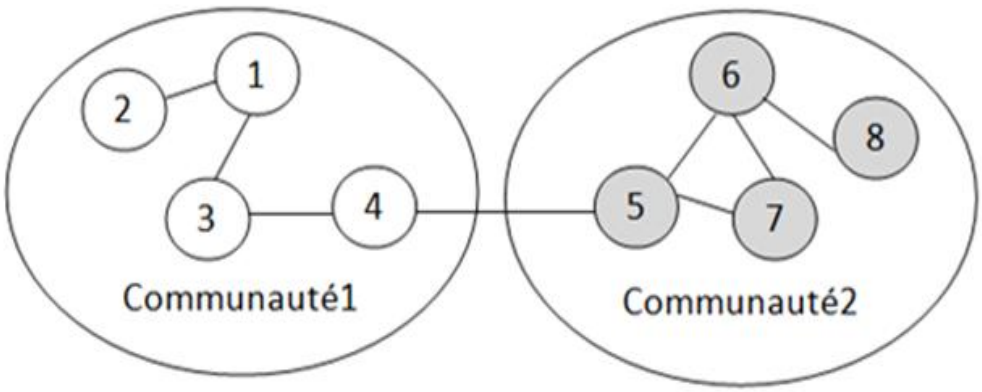


Figure II.5. Exemple de partitionnement de graphe de lien en communautés disjointes.

c) Déduction de communautés chevauchantes

Après avoir partitionné le graphe de lien G' en communautés disjointes, 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 des arêtes 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 nœuds), obtenant à partir des communautés disjointes (pour le graphe de lien), sont présentées dans la figure suivante:

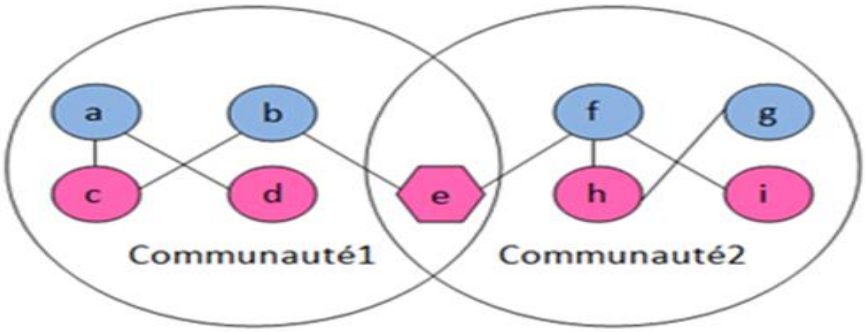


Figure II.6. Communautés chevauchantes du graphe biparti.

Conclusion

On a présenté dans ce deuxième chapitre, des objectifs et quelques domaines d'application de chevauchement des communautés dans les réseaux, puis on a introduit la notion de détection de communautés chevauchantes, suivi des méthodes utilisées pour la détection de communautés chevauchantes dans les réseaux.

Après avoir terminé de détecter les communautés chevauchantes, il est indispensable de mesurer la qualité du partitionnement obtenu, pour évaluer ce dernier, des mesures de qualité ont été développées, une de ces fonctions la plus utilisée est la modularité.

Pour maximiser la modularité, des méthodes d'optimisation de la modularité pour une décomposition en communautés chevauchantes ont été conçu. Le chapitre suivant traitera de cette notion.

Chapitre III

Optimisation de la modularité chevauchante

Introduction

L'objectif de la détection de communauté est de découper un graphe ou réseau en communautés ayant une forte densité inter-communautés et une faible densité intra-communautés. Pour évaluer la qualité de se découpage, de nombreuses méthodes qui s'appuient sur des fonctions objectives ont été utilisées. Parmi ces fonctions, on trouve le critère de modularité qui a été largement utilisée dans les études récentes.

Le but du présent chapitre est de définir la notion de la modularité, en particulier celle de Newman et celle de Mancoridis, et l'adaptation de cette dernière aux graphes bipartis. En citant par la suite, quelques méthodes d'optimisation de la modularité.

I. Evaluation de la qualité du partitionnement d'un réseau en communautés

Dans le but d'évaluer la qualité du partitionnement d'un réseau en communautés, des fonctions de qualité ont été développées. La modularité est la fonction de qualité la plus récente et largement utilisée. Elle est introduite initialement par Newman dans le but d'analyser des réseaux sociaux.

1. Définition de la modularité

La modularité est une fonction de densité des arêtes à l'intérieur des communautés d'un graphe, elle est comprise entre -1 et 1. Elle se base sur le principe qu'un bon partitionnement d'un graphe implique un nombre d'arêtes intra-communautés important et un nombre d'arêtes inter-communautés faible. Elle est donc décrite comme la proportion des arêtes incidentes sur une communauté donnée moins la valeur qu'aurait été cette même proportion si les arêtes étaient disposées au hasard entre les nœuds du graphe. [19]

Nous présentons ci-dessous la modularité de Newman, puis celle de Mancoridis, et son adaptation au graphe biparti.

Ø La modularité de Newman

Cette modularité a été largement utilisée dans les études récentes. Elle représente la métrique de qualité pour l'évaluation du partitionnement d'un réseau en communautés. [3,9]
Son principe est le suivant :

- Posons, pour un nœud v, c_v la communauté de v.

- Posons m le nombre total de lien du graphe G, tel que :

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

Où : 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 :

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

Dont:

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

$£(c_w, i)$: Cette écriture veut dire que le nœud w appartient à la communauté i.

Ou de manière plus intuitive : e_{ii} est le poids (normalisé) des arêtes dans la communauté i.

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

Où $k_v = \sum_{\mathrm{w}} \mathrm{A_{vw}}$ ou formulé autrement, k_v est le degré de v.
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, ceux-ci nécessitent moins de calcul, ce qui les rend simples et utiles dans les réseaux réels de grande taille.

La complexité temporelle de ces algorithmes est O(n) ou n est le nombre d'arêtes dans le graphe. [10]

Ø Modularité de Mancoridis

La modularité MQ (Modularity Quality), est une mesure introduite par Mancoridis [18] pour mesurer la qualité d'une partition de nœuds d'un 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.

Ø Modularité de Mancoridis pour des 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 :

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

Ø 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 MQ_{Over} adaptée par Bourqui [23] MQ_{Over} est définie comme la différence entre la cohésion interne et la cohésion externe des groupes comme suit :

$$\mathbf{MQ_{Over} = MQ^{+} - MQ^{-}_{Over}} \qquad \mathbf{(2)}$$

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

Où :

$S(C_i, C_j) = \frac{E(C_i, C_j)}{|C_i|*|C_j|}$. 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 C_i et C_j.

Et :

$$S_{Over}(Cm_i, Cm_j) = \frac{E(Cm_i, Cm_j \backslash (Cm_j \cap Cm_i))}{|Cm_i| * |Cm_j(Cm_j \cap Cm_i)|}$$

Tel que :

$|Cm_j \backslash (Cm_j \cap Cm_i)|$ est le nombre de sommets du groupe Cm_j dépourvu des sommets qu'il a en commun avec Cm_i.

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

Ø Adaptation de MQ_{Over} à un graphe biparti

Soit G (Vr, Vb, E) un graphe bipartite 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, |E|= m et A[n, n] la matrice d'adjacence du graphe, avec A[i,j]=1 si l'arête (i, j) Є E et 0 sinon.

La mesure de qualité d'une décomposition des sommets en groupes chevauchants dans un graphe biparti sera :

$$MQB_{Over} = MQB^{+} - MBQ^{-}_{Over}$$

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 $Cm_i = Cmr_i$ U 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 :

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

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 :

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

Exemple d'illustration :

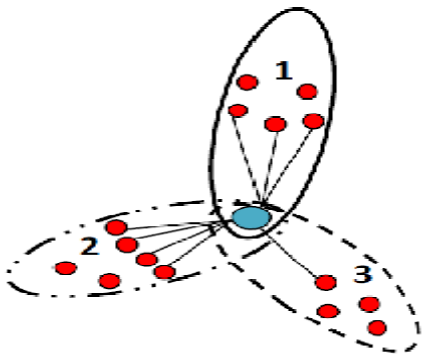


Figure III.1. Un nœud appartenant à 3 groupes mais n'appartient qu'à 2 communautés chevauchantes

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

II. Méthodes pour optimisation de la modularité

Vue son efficacité, la modularité est utilisée dans la plupart des méthodes d'optimisation récentes de détection de communautés.

Notons que, les méthodes qu'on va citer tout de suite, sont celles reconnues comme récentes et efficaces. [7]

Ø Louvain

Louvain est une méthode pour optimiser la modularité. Au départ chaque sommet est dans sa propre communautés, puis chaque sommet prend la communauté d'un de ces voisins de tel sorte que le gain de modularité soit maximal. Cette opération est répété plusieurs fois sur l'ensemble des sommets des différentes communautés jusqu'à ce qu'aucun gain ne soit possible.

Les principaux avantages de cette méthode sont sa rapidité, qui lui permet de traiter des graphes de grande taille, allant jusqu'à plusieurs milliards de liens et son excellente précision, en comparaison avec d'autres méthodes. [7,8, 22]

Ø FastGreedy

FastGreedy est une méthode agglomérative basée sur une optimisation de la modularité. En partant du partitionnement le plus fin (une communauté par nœud), les communautés sont rassemblées progressivement si cela permet d'augmenter la modularité. Deux sommets rassemblés lors d'une étape de FastGreedy appartiendront toujours à la même communauté. Un des avantages de cette méthode est sa rapidité d'exécution. [7]

Ø Algorithme de fusion

L'algorithme présenté pour maximiser la modularité des partitions strictes est constitué de deux phases. Dans la première, le point de départ est l'ensemble des singletons. Cette partition, dite atomique, est de modularité nulle puisqu'il n'y a pas d'arêtes internes. A chaque étape, et tant que la modularité croît, deux classes, telles que la partition résultante offre un gain de modularité maximum, sont réunies. L'algorithme s'arrête lorsque les classes ne peuvent plus être fusionnées sans faire décroître la modularité. Dans une seconde phase, une

procédure de transfert est appliquée qui permet d'augmenter la modularité et aboutit à une partition qui n'est plus dans la hiérarchie. [6]

III. Méthode d'optimisation de la modularité chevauchante

La plupart des méthodes d'optimisation de la modularité sont destinées pour la détection de communautés disjointes, mais, elles peuvent être adaptées la détection de communautés chevauchantes. Cependant, peu sont les méthodes directes destinées pour l'optimisation de la modularité d'un découpage en communautés chevauchantes.

- **CFinder**

Cette méthode est basée sur une recherche de motifs locaux. 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 chevauchantes, un sommet pouvant appartenir à plusieurs k-cliques non forcément adjacentes. C'est la seule méthode évaluée ici qui permette un recouvrement des communautés. Une limite de cet algorithme est qu'il nécessite un paramétrage : la valeur de k (la taille des communautés à considérer). [7]

- **Notre algorithme :**

Dans la méthode qui nous été donnée, nous partons d'un partitionnement de l'ensemble des arêtes d'un graphe biparti. Ce partitionnement nous définit une décomposition des nœuds (sommets) en groupes chevauchants (ce partitionnement représente nos données d'entrée) et nous allons appliquer à cette décomposition un algorithme pour optimiser la modularité recouvrante.

Soit O_{Over} l'ensemble des groupes *Cmi* constitués exclusivement de nœuds chevauchants.

L'optimisation se fera par le filtrage des nœuds de la décomposition d'entrée:

Nous optimiserons la modularité MQB_{Over} localement en supprimant les groupes *Cmi* de l'ensemble O_{Over}, qui diminuent la valeur de la modularité. En effet les groupes pour lesquels nous avons $S(Cm_i, Cm_i) < \sum_{j \neq i} S_{Over}(Cm_i, Cm_j)$ diminuent la modularité. Après la suppression d'un groupe, certains nœuds peuvent changer de statut (un nœud chevauchant devient un nœud non chevauchant). On met alors à jour les nœuds des groupes qui sont en relation avec le groupe supprimé.

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 doivent vérifier les nœuds chevauchants 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.

L'algorithme est récapitulé par les étapes suivantes :

1 **Pour** tout $Cm_i \in O_{Over}$ calculer $mq_i = S(Cmi, Cmi) - \sum_{j \neq i} S_{Over}(Cm_i, Cm_j)$ **fait;**

2 **S**'il existe des groupes tels que $\mathbf{mq_i} < 0$ **alors** supprimer le groupe ayant le plus petit $\mathbf{mq_i}$ **sinon** aller à 5

3 Mettre à jour $\mathbf{O_{Over}}$ et les autres groupes

4 **aller à 1**

5 **pour** tout *i* chevauchant **faire** *// filtrage des nœuds chevauchants*

pour tout $Cm_j \in Ov_i$ **faire**

Si $[i \in Cmr_j$ et $\frac{E(i,Cmj)}{|Cmbj|} < Min\left((\sum_j \frac{E(i,Cmj)}{|Cmbj|})/\,|Ov_i|,\ o.5\right)]$ **ou**

$[i \in Cmb_j$ et $\frac{E(i,Cmj)}{|Cmrj|} < Min\left((\sum_j \frac{E(i,Cmj)}{|Cmrj|})/\,|Ov_i|,\ o.5\right)]$

alors supprimer i de la communauté Cm_j

fsi

fait ;

fait ;

6 Suppression des communautés à 1ou 2 nœuds

Pour tout $Cm_i \in Cm$ *telle que* $|Cm_i|=1$ ou $|Cm_i|=2$ **faire**

pour tout *n* non chevauchant $\in Cm_i$ **faire**

Placer n dans les communautés du voisin qui a donné une meilleure modularité

fait ;

pour tout *n* chevauchant Є Cm_i, **faire**
Supprimer le nœud n de la communauté Cm_i
fait ;
fait ;
Fin

Conclusion

Nous avons expliqué dans ce présent chapitre la notion de modularité, en particulier celle de Newman et celle de Mancoridis. Nous avons vu aussi l'adaptation de cette dernière à un graphe biparti. Par la suite, des fonctions objectives et des méthodes d'optimisation de la modularité, ont été présentées.

Nous présenterons dans le chapitre suivant la conception et l'implémentation de la méthode qui nous a été donnée dans un environnement purement informatique afin de pouvoir tester sa performance et son efficacité.

Chapitre IV

Conception et implémentation

Introduction

Notre travail consiste à implémenter un algorithme de filtrage des nœuds pour optimiser la modularité de Mancoridis pour une décomposition des nœuds en groupes (communautés) chevauchants, qui est le résultat d'un partitionnement de l'ensemble des arêtes d'un graphe biparti.

Dans ce chapitre nous allons commencer par le codage des données à manipuler, à savoir la représentation du graphe biparti et le fichier contenant les communautés chevauchantes et les structures de données employées. Une fois que ces données sont définies, nous présenterons l'organigramme de l'application et les différents algorithmes implémentés pour réaliser notre logiciel.

I. Codage des données

1. Format d'introduction du graphe biparti et la décomposition de groupes chevauchants

Ø Format d'introduction du graphe biparti

Dans un graphe biparti, il est impossible d'avoir un lien entre deux nœuds du même type, à cet effet nous avons représenté les nœuds d'une famille par des indices colonnes, et les nœuds de l'autre famille par des indices lignes. De même, les liens entre les nœuds sont représentés par le contenu des cellules. Cette représentation évite le gaspillage d'espace mémoire.

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) et la deuxième indique le nombre de nœuds de la deuxième famille (nombre de colonne).

- *Les autres lignes contiennent les couples de nœuds qui se relient*. C.à.d. qu'au lieu de mettre toute la matrice qui représente le graphe dans un fichier, on insère seulement le couple de nœuds qui sont reliés sur chaque linge. Le nombre total de couples représentés sur le fichier, indique le nombre total de liens existant dans le graphe.

La figure IV.1 représente une capture d'écran d'un fichier sur lequel nous avons sauvegardé un graphe biparti.

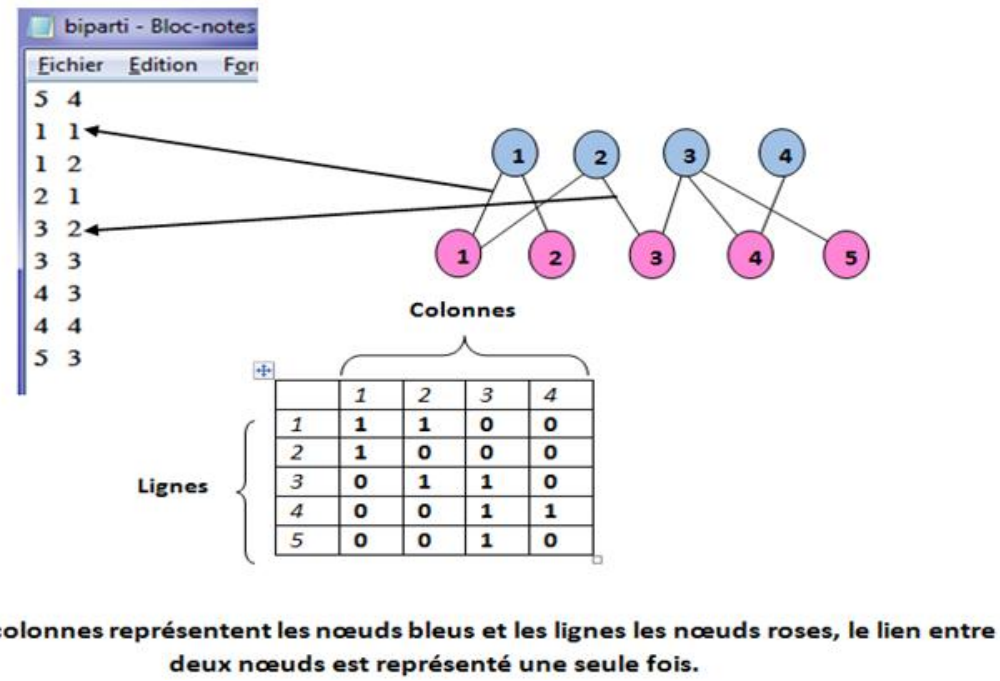


Figure IV.1. Représentation d'un graphe biparti dans un fichier.

Ø Représentation du fichier de la décomposition

Le fichier de la décomposition est constitué d'une séquence de groupes (communautés) de nœuds, représentés par leurs identifiant dans le graphe biparti, séparés par des point virgules (;).

Notons que cette décomposition de communautés est le résultat d'un partitionnement de l'ensemble des arêtes d'un graphe biparti, d'où leur chevauchement.

La Figure IV.2 représente une capture d'écran d'une décomposition de nœuds en groupes chevauchants.

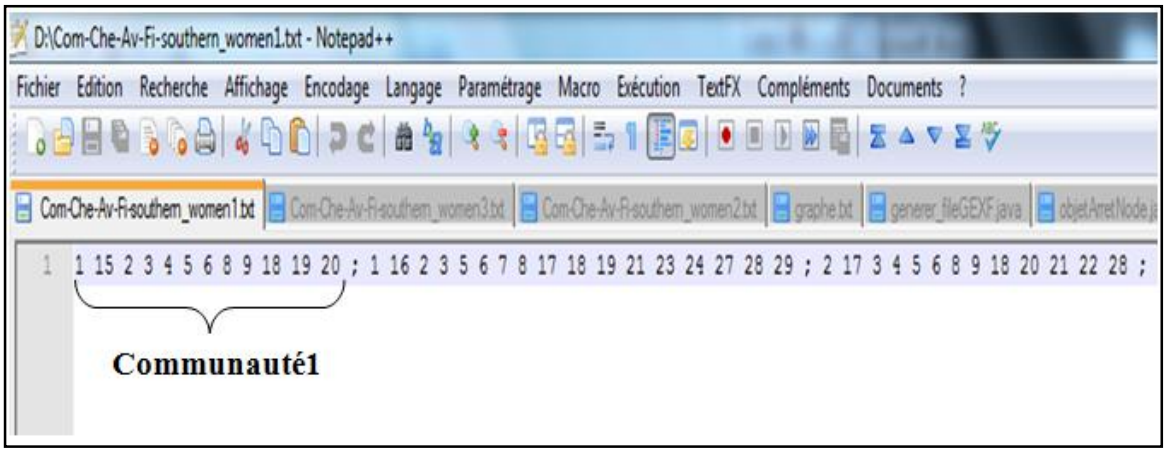


Figure IV.2. Le fichier de la décomposition.

2. Les structures de données

Ø ***Communauté*** est un objet composé de deux éléments : le premier est un entier (IDcomm), pour indiquer l'identifiant de chaque communauté et le deuxième (ListNoeud), est un tableau dynamique de type entier qui dispose les nœuds appartenant à cette communauté.

IDcomm	ListNoeud

Ø ***GroupecommunauteChevauchante* :** est un tableau dynamique employé pour structurer chaque groupe de nœuds sous forme de communautés.

Cette représentation en communautés facilite le calcule de la modularité et la génération du fichier qui permet de visualiser le graphe après le filtrage des nœuds.

Communauté1	Communauté2	. . . . .	Communauték

Ø ***O_{over} (Ensemble de communautés constituées exclusivement des nœuds Chevauchants)* :** est un tableau dynamique, qui contient les communautés constituées que des nœuds chevauchants.

Ø ***Ovi (ensemble de nœuds chevauchants) :*** est un tableau dynamique qui contient les nœuds chevauchants.

II. Etapes de l'application

Après avoir présenté les structures de données employées afin de coder les variables nécessaires pour la réalisation de notre programme, nous allons aborder dans cette partie les différentes étapes de notre application et les différents algorithmes implémentés pour la réalisation de notre logiciel.

1. L'organigramme

Les procédures principales que nous avons implémentées et les interactions entre elles sont représentées dans l'organigramme ci-dessous

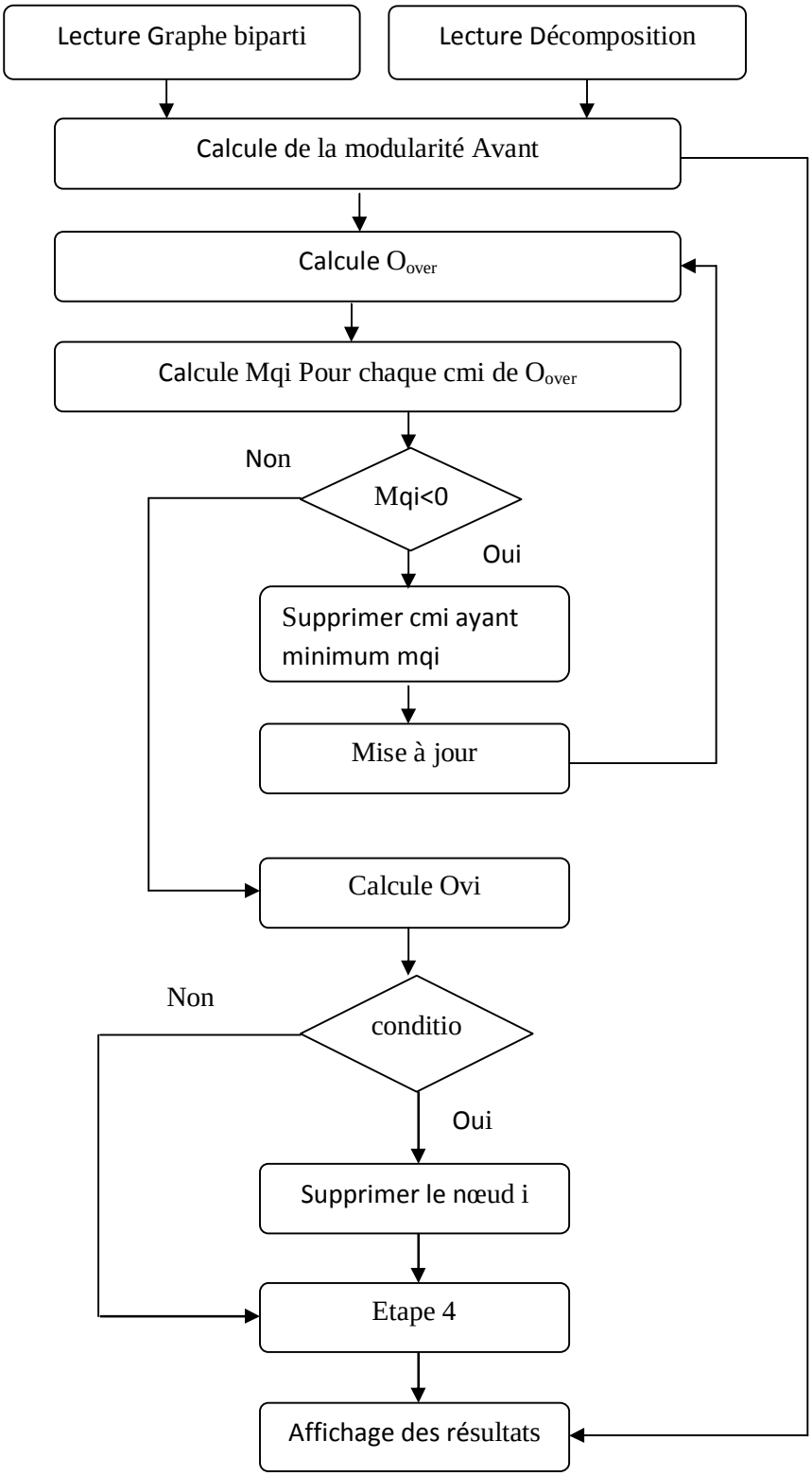


Figure IV.3. Organigramme de l'application.

L'organigramme est constitué de 5 principales étapes :

- ***Etape 1*** : dans cette étape, on lit les deux fichiers d'entrée, celui du graphe biparti pour générer la matrice de liens et celui de la décomposition pour générer le groupe de communautés chevauchantes.

- ***Etape 2*** : dans cette étape, on calcule la modularité du graphe initiale afin de comparer le résultat avec celui qu'on aura après l'application de l'algorithme. Pour chaque communauté exclusivement chevauchante on calcule la valeur de la modularité (mqi). S'il existe des communautés ayant mqi<0 alors on les supprime et on met jour le groupe de communautés chevauchantes.

- ***Etape 3*** : dans cette étape, on retire le groupe de nœuds chevauchants (Ovi), puis on supprime chaque nœud de ce groupe qui vérifie l'une des conditions suivantes :

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

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

- ***Etape 4*** : après avoir filtré les nœuds, on a appliqué la dernière étape de l'algorithme qu'est la suppression de communautés ayant un ou deux nœuds et placer ces derniers dans la communauté voisine qui donne la meilleure modularité. Ensuite, on calcule la modularité pour le nouveau groupe de communauté obtenu.

- ***Etape 5*** : cette étape est réservée pour l'affichage de la valeur de la modularité et le nombre de communautés trouvées.

2. Les algorithmes

2.1. Lecture du graphe biparti

Pour la lecture du graphe à partir du fichier texte et la construction de la matrice correspondante, nous avons programmé la procédure suivante :

Algorithme 1 :

Début

Ouvrir en écriture le fichier (monfichier)

Lire la première ligne

NbrNoeudBleu ⟵ le premier élément de la ligne

NbrNoeudRouge ⟵ le premier élément de la ligne

Création de la matrice [NbrNoeudBleu][NbrNoeudRouge]

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-1][colonne-1]⟵1

fin ;

Fin.

2.2. Lecture du fichier de la décomposition

La lecture du fichier de la décomposition à partir du fichier texte est illustrée dans l'algorithme ci-dessous:

Algorithme 2 :

Début

Ouvrir en écriture le fichier (mon fichier)

Créer GroupCommunauteChevauchante()

Créer listeNoeud()

Entier idcommunaute ⟵ 0 ;

***Tant que** (le fichier n'est pas vide) faire*

début

Lire la première ligne ;

Tok le premier token du fichier ;

Pour** (i allons de 1 jusqu'à n) **faire

Si** (tok ≠ " ;") **alors

Ajouter le token à listeNoeud ;

Sinon

Si** (tok =' ;') **alors

Ajouter la communaute à GroupCommunauteChevauchante

id_communaute id_communaute +1 ;

fin si;

fin sinon ;

Fin si ;

Fin.

2.3.Evaluation de qualité de communautés chevauchantes (Modularité)

Pour évaluer la qualité du partitionnement en communautés chevauchantes, nous utilisons la modularité de Mancoridis MQB_{Over} présentée dans le chapitre précédant.

Avant de présenter l'algorithme que nous avons programmé, nous donnons d'abord des explications sur les variables utilisées.

taille: le nombre de communautés chevauchantes

nbrLienInterne : est le nombre de liens interne pour une communauté donnée.

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 rouge de la première famille.

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

Algorithme 3

Début

Initialiser MQBover, MQBplus et MQBmoins à zéro

Pour *(i allons de 0 jusqu'à taille-1)* ***faire***

début

Initialiser bnrBleu et nbrRouge à zéro

Initialiser nbrLienInterne à zéro

Pour *chaque communauté du GroupCommunauteChevauchante* ***faire***

Pour *chaque nœud de cette communauté* ***faire***

début

Si *le nœud appartient aux nœuds bleus* ***alors***

nbrBleu + +

sinon

nbrRouge + + ;

pour** chaque voisin de nœud **faire

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

nbrLienInterne + + ;

fin si ;

nbrLienInterne ⟵ nbrLienInterne/2

*MQBplus⟵ MQBplus + (nbrLienInterne/ nbrBleu * nbrRouge)*

Fin

MQBplus ⟵MQBplus/ nbr de communauté

Pour** chaque communauté1 de GroupCommunauteChevauchante **faire

pour** chaque communauté2 de Over **faire

si** (communauté1.id_communaute = communauté2.id_communauté) **alors

début

initialiser nbrBleu1, nbrRouge1, nbrBleu2 et nbrRouge2 à zéro

initialiser nbrLienInterne à zéro

éliminer de la communauté2 tous les nœuds en commun entre communauté1 et communauté2 et insérer ces nœuds dans listCommun

pour** chaque noeud1 de communaute1 **faire

pour** chaque noeud2 de communauté2 **faire

si** noeud1 est un voisin de noeud2 **alors

nbrLienExterne + +

calculer le nombre de nœud bleu de la communauté1 :nbrNoeudBleu1

calculer le nombre de nœud rouge de la communauté1 :nbrNoeudRleu1

calculer le nombre de nœud bleu de la communauté2 :nbrNoeudBleu2

calculer le nombre de nœud rouge de la communauté2 :nbrNoeudBleu2

*MQBmoins ⟵ MQBmoins + (nbrLienExterne / (nbrRouge1 * nbrBleu2 + nbrRouge2 * nbrBleu1))*

Réinitialiser dans communauté2 tous les nœuds ce trouvant dans listCommun

Vider listCommun

fin ;

MQBmoins⟵MQBmoins/(taille)(taille-1)

MQBover⟵MQBplus – MQBmoins

Fin.

2.4. Détection de communautés constituées exclusivement des nœuds chevauchants (Oover)

Algorithme 6 :

Debut

Créer liste groupcommunateEXchevauchante() ;

Pour** (i allons de 0 jusqu'à taille GroupCommunauteChevauchante-1) **faire

Pour** chaque communaute1 (i) de GroupCommunauteChevauchante **faire

Pour** chaque noeud1 de communaute1 (i) **faire

Si** (chaque nœud de communaute1 appartient à liste noeudchevauchant) **alors

Ajouter communaute1 à groupcommunateEXchevauchante() ;

fsi ;

Fin.

2.5. Calcule de la valeur la modularité de chaque communauté (mqi)

Algorithme 5 :

Début

Initialiser Scmi , Sover à zéro

Pour** (i allons de 0 jusqu'à taille-1) **faire

Début

Initialiser bnrBleu et nbrRouge à zéro

Initialiser nbrLienInterne à zéro

Pour** chaque communauté du Oover **faire

Pour** chaque nœud de cette communauté **faire

Début

Si** le nœud appartient aux nœuds bleus **alors

nbrBleu + +

sinon

nbrRouge + + ;

pour** chaque voisin de nœud **faire

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

nbrLienInterne + + ;

finsi ;

nbrLienInterne ⟵ nbrLienInterne/2

*Scmi ⟵ Scmi (nbrLienInterne / nbrBleu * nbrRouge)*

fin ;

Retourner Scmi pour chaque communauté de Oover

fin ;

Pour *chaque communauté1 de Oover* ***faire***

pour *chaque communauté2 de Oover* ***faire***

si *(communauté1.id_communaute = communauté2.id_communauté)* ***alors***

début

initialiser nbrBleu1, nbrRouge1, nbrBleu2 et nbrRouge2 à zéro

initialiser nbrLienInterne à zéro

éliminer de la communauté2 tous les nœuds en commun entre communauté1

et communauté2 et insérer ces nœuds dans listCommun

pour *chaque noeud1 de communaute1* ***faire***

pour *chaque noeud2 de communauté2* ***faire***

si *noeud1 est un voisin de noeud2* ***alors***

nbrLienExterne + +

calculer le nombre de nœud bleu de la communauté1 :nbrNoeudBleu1

calculer le nombre de nœud rouge de la communauté1 :nbrNoeudRleu1

calculer le nombre de nœud bleu de la communauté2 :nbrNoeudBleu2

calculer le nombre de nœud rouge de la communauté2 :nbrNoeudBleu2

*Sover⟵Sover + (nbrLienExterne /(nbrRouge1 * nbrBleu2 + nbrRouge2 * nbrBleu1))*

Réinitialiser dans communauté2 tous les nœuds ce trouvant dans listCommun

Vider listCommun

fin ;

Sover⟵ Sover /(taille)

fin ;

Retourner Sover pour chaque communauté de Oover

Fin.

2.6. Calcul Ovi (ensemble de nœuds chevauchants)

Algorithme 6:

Début

Créer liste noeudchevauchant () ; // la liste qui contient les nœuds chevauchants

Pour *(i allons de 0 jusqu'à taille GroupCommunauteChevauchante())* ***faire***

Pour *chaque communaute1 (i) de GroupCommunauteChevauchante()* ***faire***

Pour *chaque noeud1 de communaute1 (i)* ***faire***

Entier k ⟵ 0 ;

Booleen exist ⟵ false ;

Tant que *(k inférieur à i)* ***faire***

Pour *chaque communaute2 (k) de GroupCommunauteChevauchante()* ***faire***

Entier h ⟵ 0 ;

Tant que *(h inférieur à taille de liste de nœuds de communate2 (k))* ***faire***

Pour *chaque nœud2 de communaute2 (k)* ***faire***

si *(nœud1=soeud2)* ***alors***

exist ⟵ true ;

fsi ;

h++ ;

si *(exist= true et noeud1 n'appatient pas à noeudchevauchant)* ***alors***

ajouter noeud1 à noeudchevauchant ;

fsi ;

k++ ;

Fin.

2.7. Génération du fichier.gexf

En plus de l'affichage des résultats de l'optimisation de la modularité chevauchante, 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 format xml pour représenter le graphe. Ce fichier se compose de deux parties : l'entête et 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 : graph, 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.

Dans notre cas, c'est les arêtes de même communauté qui ont la même couleur, par contre les nœuds sont coloriés par rapport à leurs familles excepté les nœuds qui se chevauchent (auront une couleur différente) et nous avons attribué une même taille pour tous les nœuds d'une même communauté.

Notons que la manière de générer ce fichier est identique avant et après l'application de l'algorithme, pour cela nous avons employé une seule procédure de génération de fichier (.gexf), définie par l'algorithme suivant :

Algorithme 7 :

Debut

Ouvrir en ecriture le fichier ("commChev")

Ecrire dans le fichier ("< ?xml version=\"1.0\ "encoding=\ "UTF_8\" ?>")

Ecrire dans le fichier ("<gexf xmlns

=\"http://www.gexf.net/1.2draft\"xmlns :viz

=\"http://www.gexf.net/1.2draft/viz">")

Ecrire dans le fichier ("\t < graph defautedgetype =\"undirected\" > ")

Ecrire dans le fichier ("\t\t < nodes > ")

Pour** (i allons de 0 à nbrnoeud-1) **faire

Ecrire dans le fichier ("\t\t\t < node id = \ "" + i + "\" label =\"" + i + "\"/ >")

Ecrire dans le fichier ("\t\t </nodes > ")

Ecire dans le fichier ("\t\t < edges > ")

Initialiser *idedge à zéro*

pour** (i allons de 0 à nbrnoeud-1) **faire

pour** (j allons de 0 à nbrnoeud-1)**faire

s'**il ya un lien entre i et *j **alors

début

("\t\t\t < edge id =\"" + idedge + "\" source

= \"" + i + "\" target = \"" + j + "\"/ >")

Idedge++

fin

Ecrire dans le fichier ("\t\t </edges > ")

Ecrire dans le fichier ("\t < /graph >")

Ecrire dans le fichier (" </gexf > ")

Fermer le fichier

Fin.

Conclusion

Après avoir présenté les différentes structures de données employées pour l'implémentation de notre algorithme, nous avons donné les étapes principales de l'application, puis nous avons détaillé quelques algorithmes employés dans le but de la réalisation de notre logiciel.

Nous présenterons dans le chapitre suivant en plus, de l'environnement de développement, les résultats des tests effectués sur notre algorithme de filtrage et d'optimisation afin d'évaluer ses performances.

Chapitre V

Mise en œuvre et expérimentation

Introduction

Apres avoir défini les déférentes étapes suivies pour l'implémentation de notre algorithme de filtrage et d'optimisation, nous présenterons dans ce dernier chapitre la mise en œuvre de notre logiciel.

Ce chapitre est subdivisé en deux sections, la première décrit la plate forme de développement utilisée, et la deuxième a pour but de présenter l'architecture générale de notre application ainsi que les résultats expérimentaux que nous avons obtenus.

I. Environnement de développement

Pour l'implémentation des différents algorithmes vus dans le chapitre précédant, nous avons opté pour le langage java comme outil de développement dans un environnement Windows, et les résultats du partitionnement du graphe obtenus par notre application sont visualisés à l'aide du logiciel Gephi (version Gephi 0.8 alpha).

1. Le langage Java

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

Il possède un certain nombre de caractéristiques qui ont largement contribué à son énorme succès, dont on cite :

- **Java est interprété :** 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.

- **Java est 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.

- **Java est 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,...).

- **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. Eclipse

Eclipse est un environnement de développement intégré (Integrated Development Environment), développé en Java par I.B.M, dont le but est de fournir une plate-forme modulaire pour permettre de réaliser des développements informatiques.

L'adoption d'éclipse par une large communauté de développeur est du à plusieurs raison :

- Support de plusieurs plate-formes d'exécution : Windows, Linux, Mac OS X, ...

- Une plate-forme ouverte pour le développement d'applications et extensible grâce à un mécanisme de plug-ins.

- Plusieurs versions d'un même plug-in peuvent cohabiter sur une même plate-forme.

- Un support multi langage grâce à des plug-ins dédiés : Cobol, C, PHP, …
- La plate-forme est entièrement internationalisée dans une dizaine de langue sous la forme d'un plug-in téléchargeable séparément.
- Propose des mécanismes pour développer de nouveaux plug-ins.

3. 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...
- Plugins communautaires permettant d'étendre facilement les fonctionnalités
- Tableur adapté aux données de réseaux pour combiner visualisation et analyse statistique
- Editeur de cartographie vectorielle
- Filtrage
- Support des réseaux dynamiques et des graphes hiérarchiques
- Toolkit dont les APIs permettent de construire d'autres applications

Les deux premières fonctionnalités correspondent à nos besoins :

Ø Visualisation :

Nous a permis de mieux manipuler les nœuds de différentes communautés avec ces divers éléments :

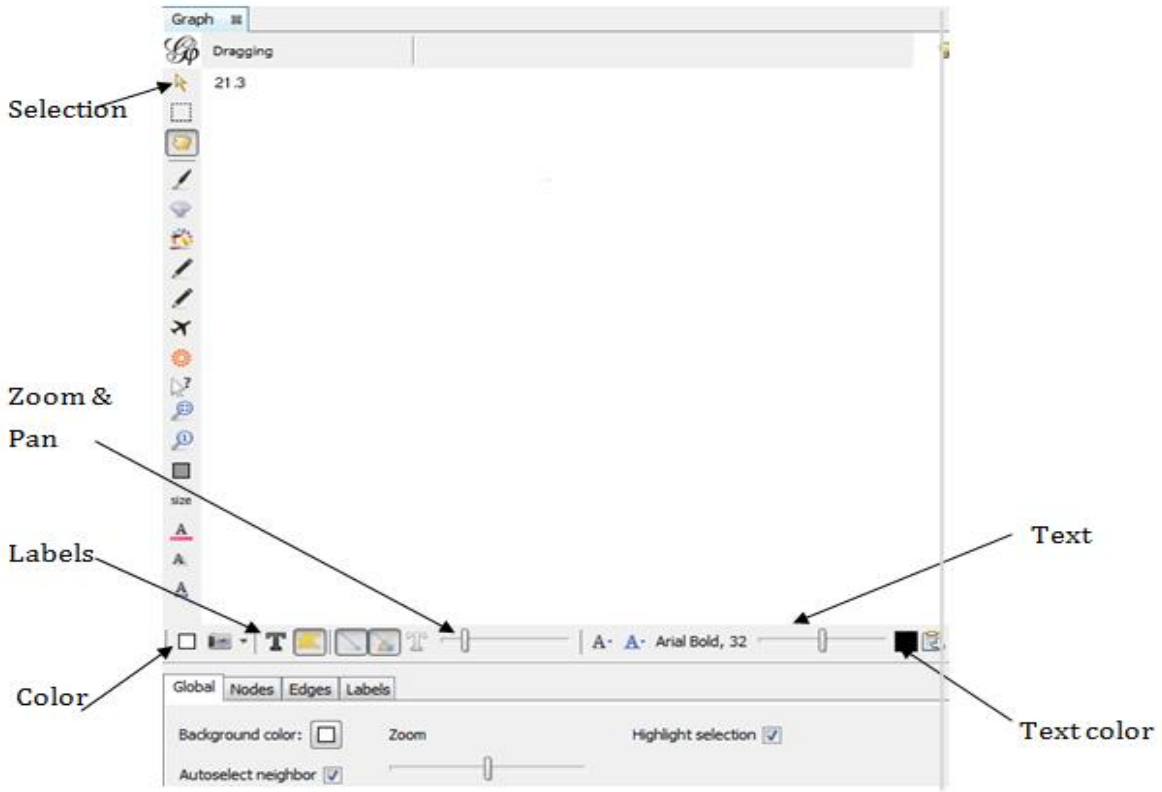


Figure V.1. Vue générale de l'interface de visualisation de Gephi

Ø Layout :

Nous avons choisi d'appliquer l'algorithme Force Atlas car il correspond à nos besoins. Il permet de rapprocher les nœuds de même taille (lors de la programmation, nous avons affecté la même taille aux nœuds de la même communauté au moment de la génération du fichier gexf).

La procédure est comme suite :

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».

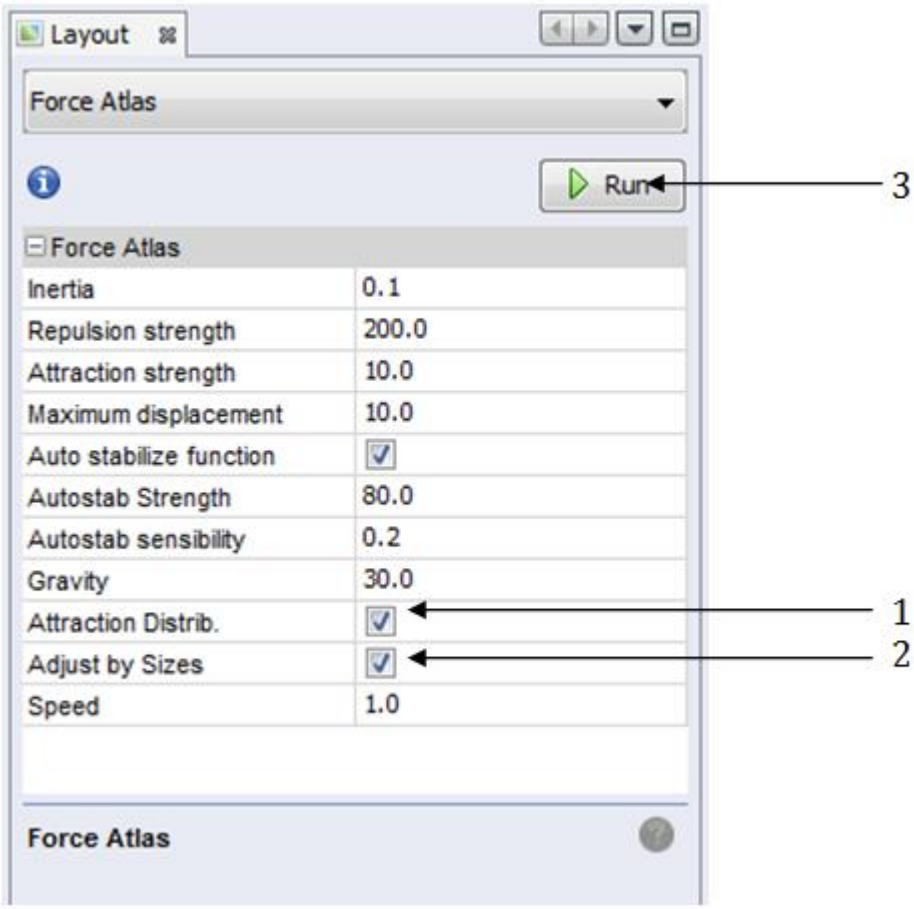


Figure V.2. Vue générale de l'interface de visualisation de Gephi

Ø Interopérabilité

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

II. Description des interfaces de notre logiciel

Ø La fenêtre principale

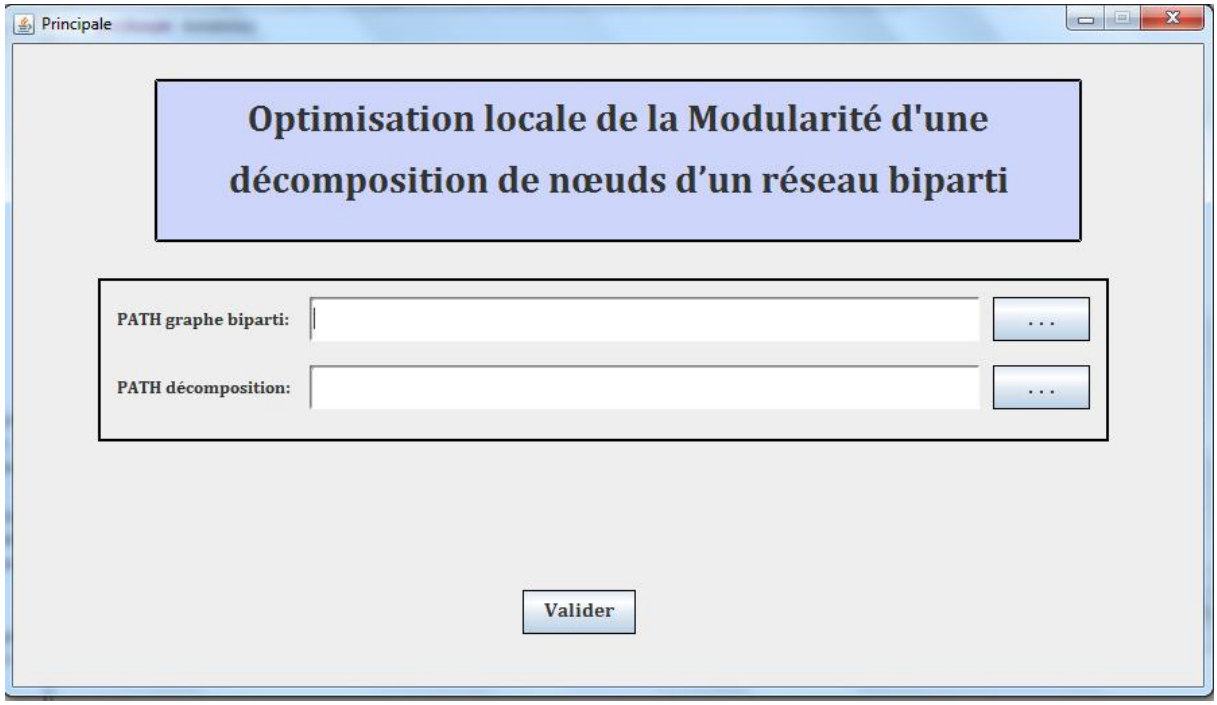


Figure V.3. La fenêtre principale

L'utilisation de cette fenêtre est comme suit :

1- Cliquer sur les boutons « … » afin de charger les fichiers d'entrée (qui contiennent la description du graphe biparti et la décomposition de communautés) sur lesquels notre algorithme de filtrage et d'optimisation sera appliqué et leurs emplacements sera affiché dans la zone « File PATH ».

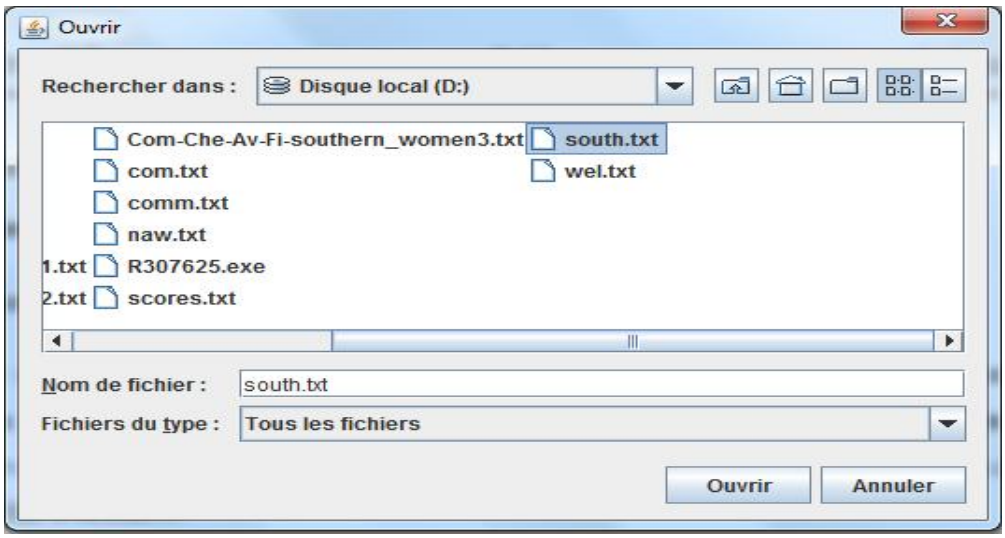


Figure V.4. Fenêtre de chargement du PATH

2- Cliquer sur le bouton « Exécuter », ce qui va lancer l'exécution de l'algorithme du filtrage et d'optimisation.

Les résultats sont retournés par la fenêtre suivante :

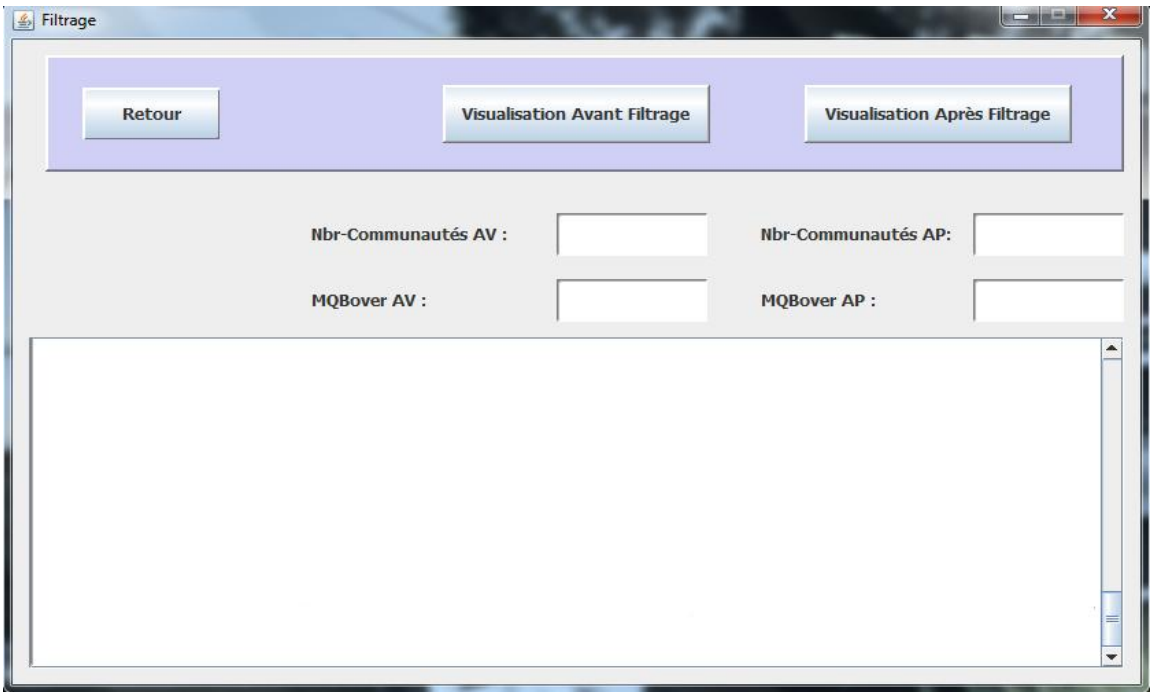


Figure V.6. Fenêtre d'exécution de l'algorithme

Remarque :

Pour lancer l'exécution de notre algorithme, l'utilisateur doit remplir les deux chemins des fichiers d'entrée, sachant que l'extension de ces derniers doit être de .txt, sinon fenêtre d'erreur suivante sera générée :

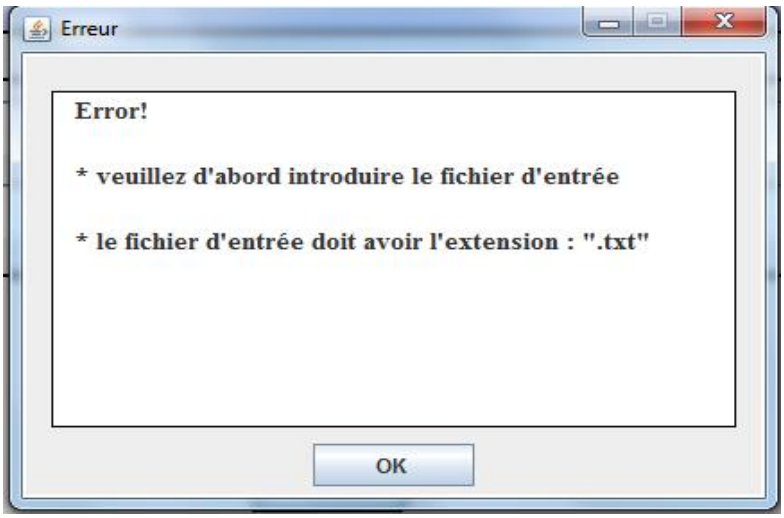


Figure V.7. Interface d'erreur.

III. Résultats expérimentaux

1. Réseau réel

Nous avons appliqué notre algorithme de filtrage et d'optimisation sur plusieurs décompositions du graphe **southern_women** (Les femmes du Sud), qu'est un réseau social comportant 18 nœuds femmes et 14 nœuds événements, avec 89 liens 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 dans le cadre d'une étude approfondie de classe et de race dans le Sud des USA.

La figure suivante représente une capture d'écran de ce réseau avant l'application de notre algorithme.

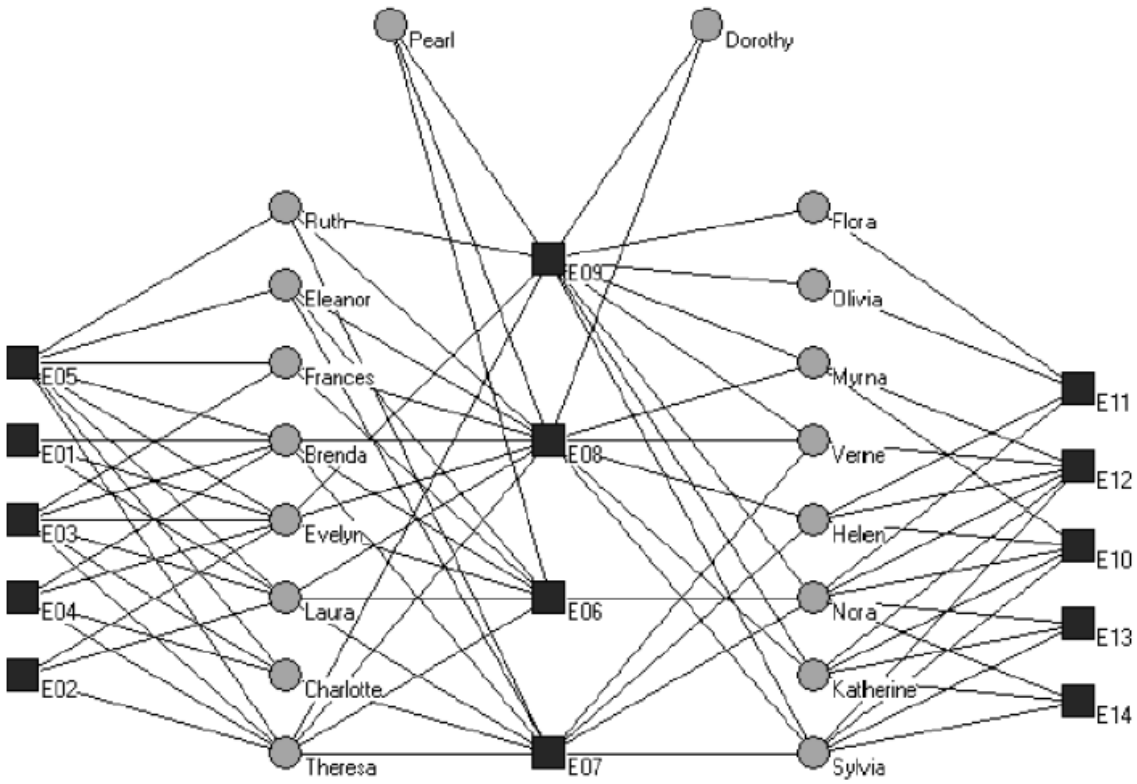


Figure V.8. Le réseau Southern-women.

Nous avons appliqué notre algorithme sur trois décompositions du graphe southern women, les résultats obtenus sont décrits dans le tableau suivant :

Graphe southern women	**Avant filtrage**		**Après filtrage**	
	Nbr communautés	MQBover	Nbr communautés	MQBover
Décomposition 1	09	0,12	06	0.50
Décomposition 2	12	0,38	06	0.47
Décomposition 3	07	0,24	07	0.32

Tableau V.1. Résultats expérimentaux sur réseau réel.

Les figures suivantes représentent des captures d'écran de la décomposition 1 du graphe southern women avant et après le filtrage.

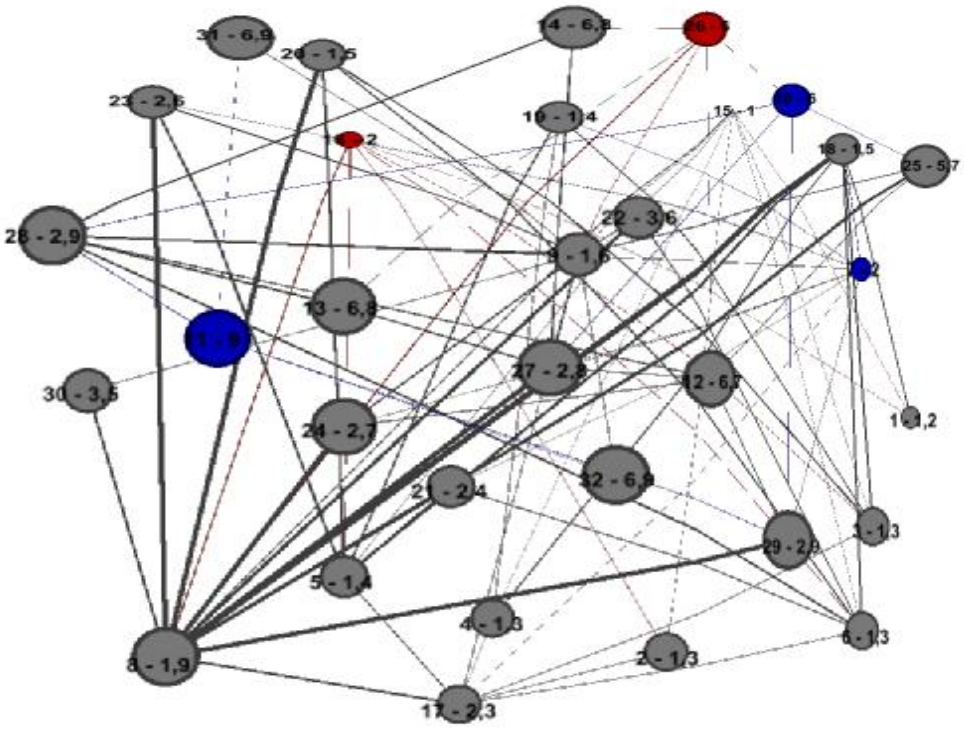


Figure V.9. La décomposition 1 avant filtrage.

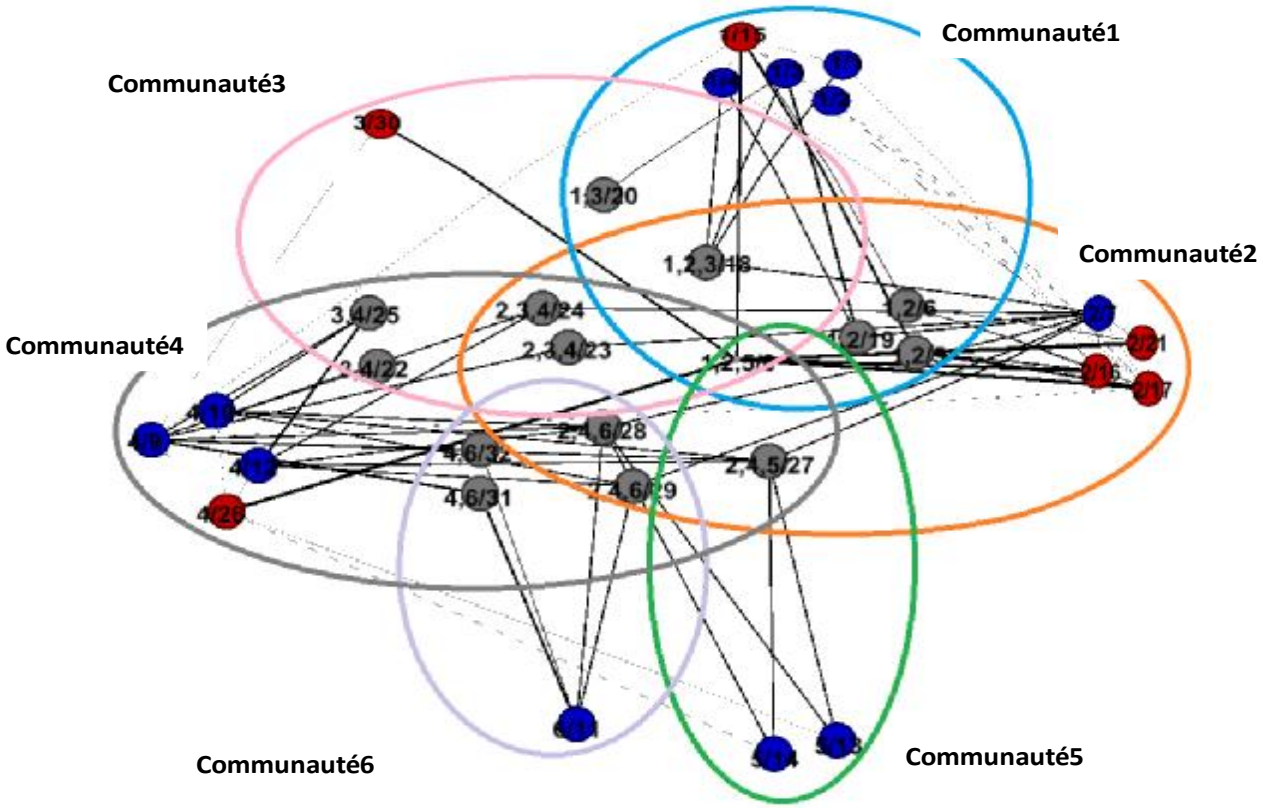


Figure V.10. La décomposition 1 Après filtrage.

2. Réseau synthétique

Nous avons appliqué notre algorithme sur trois décompositions du réseau synthétique de la figure suivante :

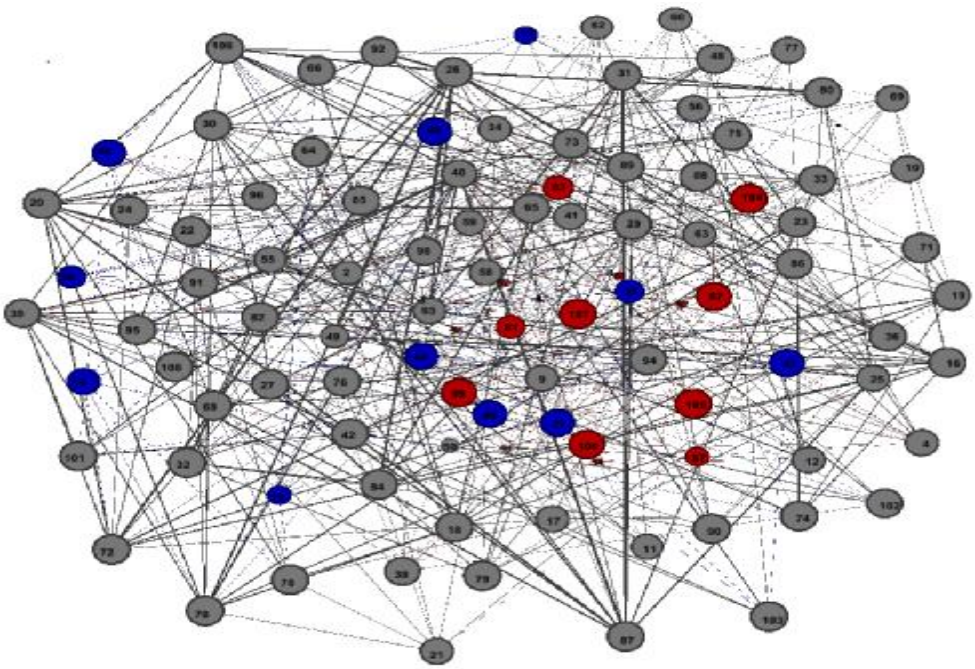


Figure V.10. Réseau synthétique avant filtrage.

Les résultats obtenus sont décris dans le tableau ci-dessous :

Graphe synthétique	**Avant filtrage**		**Après filtrage**	
	Nbr communautés	MQBover	Nbr communautés	MQBover
Décomposition 1	12	0,14	08	0.32
Décomposition 2	11	0,18	08	0.45
Décomposition 3	13	0,10	06	0.49

Tableau V.2. Résultats expérimentaux sur graphe synthétique.

Les figures suivantes représentent des captures d'écran de la décomposition 3 du graphe synthétique précédent avant et après le filtrage.

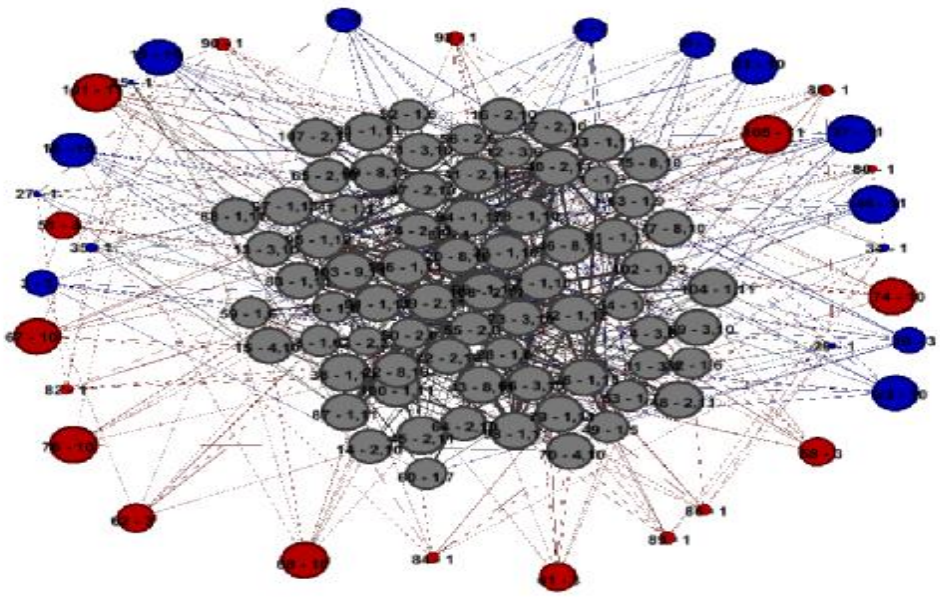


Figure V.11. La décomposition3 avant filtrage.

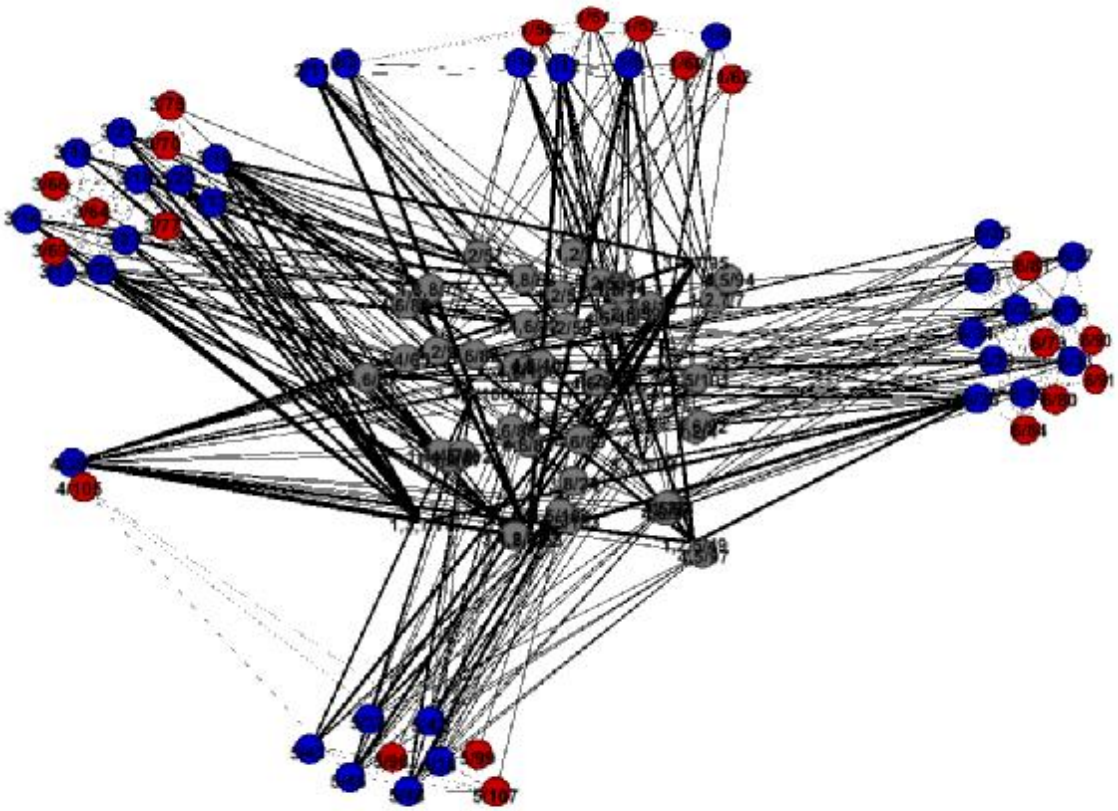


Figure V.12. La décomposition 3 après filtrage.

Conclusion

Ce chapitre termine notre étude pratique sur l'implémentation de l'algorithme de filtrage et d'optimisation. Nous avons constaté une optimisation importante de la modularité de Mancoridis, ce qui implique une meilleure densité dans chaque communauté obtenue, donc un meilleur découpage en communautés chevauchantes.

Conclusion générale

Dans notre projet, nous avons implémenté un nouvel algorithme de filtrage et d'optimisation sur une décomposition de nœuds en groupes (communautés) chevauchants, qui est le résultat de partitionnement de l'ensemble des arêtes d'un graphe biparti.
Afin d'évaluer la qualité de ce découpage en communautés chevauchantes, l'algorithme utilise le critère de la modularité de Mancoridis adaptée aux graphes bipartis.

Après l'application de cet algorithme, nous avons constaté une optimisation de la modularité, ce qui signifie une amélioration de la structure des communautés du réseau biparti.

La détection de communautés est un domaine de recherche extrêmement actif et fécond, la course à des meilleures méthodes de détection et d'optimisation toujours continue. Certaines problématiques restent encore qu'à leurs balbutiements tel que la détection de communautés chevauchantes dans les réseaux ou graphes en général, en particulier dans les graphes biparti et les graphes orientés.

Bibliographie

[1] Charles-Edmond BICHOT, Elaboration d'une nouvelle métaheuristique pour le partitionnement de graphe : la méthode de fusion-fission. Application au découpage de l'espace aérien. Thèse de Doctorat, 2007.

[2] Pascal Pons, Algorithmique des grands réseaux d'interactions: détection de structures de communautés. Rapport de DEA, 2004.

[3] Aït Yakoub Zina et Belhadi Rabia, Détection de communautés dans les réseaux complexes en utilisant les algorithmes génétiques. Rapport du Master, Université de Tizi-Ouzou, 2011.

[4] Jean-Baptiste Angelelli, Alain Guénoche et Laurence Reboul, Détection de communautés disjointes ou chevauchantes dans les réseaux, Article.

[5] Nicolas Krummenacher, Heuristiques de conception de topologies réseaux: application aux réseaux locaux industriels, Thèse de Doctorat, 2002.

[6] Pascal Pons, Détection de communautés dans les grands graphes de terrain, Thèse de Doctorat, 2007.

[7] Emmanuel Navarro et Rémy Cazabet, Détection de communautés, étude comparative sur graphes réels. Université de Toulouse, Article.

[8] Thomas Aynaud et Jean-Loup Guillaume, Détection de communautés à long terme dans les graphes dynamiques. Université Pierre et Marie Curie, Article.

[9] Vinsenzo Nicosia, Extending the definition of modularity to directed graphs with overlapping communities. Université Catania, Italie. Article

[10] Newman, M.E.J. "Fast algorithm for detecting community structure in networks", Article Physical Review, 2004.

[11] Mustapha Oughdi, régulation de la demande dans les réseaux mobiles par optimisation de la tarification. Thèse de Doctorat, Université de Franche-Comté, décembre 2008.

[12] Clara Pizzuti, Overlapped Community Detection in Complex Networks, ICAR-CNR, Italie, Article.

[13] Steve Gregory,Finding Overlapping Communities Using Disjoint Community Detection Algorithms, Université de Bristol Angleterre. Article

[14] Nan Du, Bin Wu, Bai Wang, and Yi Wang, Overlapping Community Detection in Bipartite Networks, Université Télécommunications de chine. Article

[15] Arnau Padrol-Sureda, Guillem Perarnau-Llobet, Julian Pfeifle et Victor Muntés-Mulero, Overlapping Community Search for Social Networks. Université Pollytecnique de Catalunya. Article

[16] Nam P. Nguyen, Yilin Shen, Thang N. Dinh, On the evolution of overlapping communities in dynamic mobile networks. Article

[18] 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.

[19] Jean-Baptiste Angelelli, Alain Guénoche et Laurence Reboul, Détection de communautés, disjointes ou chevauchantes, dans les réseaux, Université Marsseille France, Article.

[20] Romain Bourqui, Paolo Simonetto, Fabien Jourdan, Un algorithme stable de décomposition pour l'analyse des réseaux sociaux dynamiques, Université Bordeaux France. Article.

[21] Mathieu Barthelemy, Graph clustering , Université de Lyon France, Article

[22]Thomas AYNAUD, Vincent D. BLONDEL, Jean Loup GUILLAUME et Renaud LAMBIOTTE, Optimisation locale multi-niveaux de la modularité, Article.

[23] R.Bourqui and D. Auber .Analysis of 4-connected Components Decomposition for Graph Visualization. Rapport.