Nouveaux algorithmes pour la détection de communautés disjointes et chevauchantes basés sur la propagation de labels et adaptés aux grands graphes.

Jean-Philippe Attal

Université / UNIVERSITÉ de Cergy-Pontoise / Paris Seine

T H È S E

pour obtenir le titre de

Docteur de
de l'Université de Cergy Pontoise
Spécialité : Sciences et Technologies de l'Information et de la Communication STIC

présentée et soutenue par

Jean-Philippe ATTAL

Nouveaux algorithmes pour la détection de communautés disjointes et chevauchantes basés sur la propagation de labels et adaptés aux grands graphes

---

date de soutenance 19 janvier 2017

Le jury est composé de :

Rapporteur	M. Arnaud MARTIN
Rapporteur	M. Guy MELANÇON
Examinateur	M. Henry SOLDANO
Directeur	M. Marc ZOLGHADRI
Examinateur	Mme. Maria MALEK
Examinateur	M. Dominique Laurent

1 Résumé

Les graphes sont des structures mathématiques capables de modéliser certains systèmes complexes. Une des nombreuses problématiques liées aux graphes concerne la détection de communautés qui vise à trouver une partition en sommet d'un graphe en vue d'en comprendre la structure. A titre d'exemple, en représentant des contrats d'assurances par des nœuds et leurs degrés de similarité par une arête, détecter des groupes de nœuds fortement connectés conduit à détecter des profils similaires, et donc à voir des profils à risques. De nombreux algorithmes ont essayé de répondre à ce problème. Une des méthodes est la propagation de labels qui consiste à ce que chaque nœud puisse recevoir un label par un vote majoritaire de ses voisins. Bien que cette méthode soit simple à mettre en œuvre, elle présente une grande instabilité due au non déterminisme de l'algorithme et peut dans certains cas ne pas détecter de structures communautaires.

La première contribution de cette thèse sera de *i)* proposer une méthode de stabilisation de la propagation de labels tout en appliquant des barrages artificiels pour limiter les possibles mauvaises propagations. Les réseaux complexes ont également comme caractéristique que certains nœuds puissent appartenir à plusieurs communautés, on parle alors de recouvrements. C'est en ce sens que la seconde contribution de cette thèse portera sur *ii)* la création d'un algorithme auquel seront adjointes des fonctions d'appartenances pour détecter de possibles recouvrements via des nœuds candidats au chevauchement. La taille des graphes est également une notion à considérer dans la mesure où certains réseaux peuvent contenir plusieurs millions de nœuds et d'arêtes. Nous proposons *iii)* une version parallèle et distribuée de la détection de communautés en utilisant la propagation de labels par cœur. Une étude comparative sera effectuée pour observer la qualité de partitionnement et de recouvrement des algorithmes proposés.

Table des matières

Remerciements

Cette thèse m'a permis d'acquérir une très grande ouverture d'esprit dans le monde de la recherche.

Je remercie Madame Maria Malek, qui m'a toujours soutenu, aidé, conseillé et protégé au cours de cette thèse. Madame Maria Malek m'a transmis son savoir et m'a initié à la recherche scientifique. Madame Maria Malek a toujours fait preuve d'une grande gentillesse à mon égard. Je n'aurais jamais pu réussir cette thèse sans elle.

Je remercie Monsieur Dominique Laurent, qui a toujours été à mon écoute et m'a soutenu. Ses conseils et sa très grande gentillesse m'ont permis d'accomplir ce travail de doctorat.

Je remercie Monsieur Marc Zolghadri pour avoir accepté de diriger cette thèse. Par ses conseils, sa grande gentillesse, et son enthousiasme, Monsieur Marc Zolghadri m'a aidé à toujours améliorer mon travail scientifique.

Je remercie Monsieur Arnaud Martin d'avoir accepté d'être rapporteur de cette thèse. Monsieur Arnaud Martin m'a beaucoup aidé à améliorer mon travail scientifique au cours des différentes conférences auxquelles nous avons assisté ensemble. Ses remarques et observations sur le manuscrit m'ont beaucoup aidé.

Je remercie Monsieur Guy Melançon d'avoir accepté d'être rapporteur de cette thèse. Ses analyses critiques et son expérience m'ont permis d'améliorer ce travail doctoral.

Je remercie Monsieur Henry Soldano d'avoir accepté d'être examinateur de cette thèse. Les questions que Monsieur Henry Soldano a posées durant les différentes conférences auxquelles nous assistions m'ont permis de faire évoluer mon travail, de l'améliorer, et d'ouvrir de nouvelles perspectives scientifiques.

Je remercie Monsieur Chrysostome Baskiotis qui m'a enseigné, lorsque j'étais étudiant, le langage Python qui me fut très utile dans la réalisation du travail doctoral.

Je remercie Monsieur Nesim Fintz pour m'avoir aidé financièrement durant cette thèse et de sa grande gentillesse.

Je remercie tous mes collègues de l'EISTI qui m'ont toujours aidé et soutenu durant mon travail doctoral.

Je remercie tous mes amis de m'avoir aidé et soutenu.

Je remercie ma famille et mes parents, qui m'ont toujours soutenu et aidé au cours de ma vie et durant ce travail doctoral. C'est à eux que je dédie ma thèse. Je n'aurais jamais réussi sans leur soutien. J'ai une pensée pour mes grands-parents, qui ont contribué à mon éducation et qui m'ont guidé dans ma vie.

Introduction

De nombreux systèmes réels sont représentés à l'aide de graphes. Des entités peuvent être modélisées par des sommets et leurs relations par des arêtes. Il existe de nombreux domaines et champs d'application utilisant la structure de graphe (graphes sociologiques, graphes de collaborations, réseaux biologiques (interactions protéine-protéine, réseaux de neurones, réseaux de gènes), réseaux sportifs, réseaux du web (graphes de pages web connectées), réseaux de transports, réseaux de co-achats,etc). Ces graphes portent le nom de *graphes de terrains* ou de *réseaux complexes*.

L'étude et l'analyse de ce genre de graphes révélent des informations sur les acteurs et expliquent la structure même du réseau. Parmi les nombreuses applications liées aux graphes, on peut citer *la détection de communautés* qui vise à trouver au sein d'un graphe des groupes de noeuds fortement connectés entre eux et faiblement avec le reste du graphe. La détection de communautés permet de trouver les individus d'une population les plus similaires en fonction de leurs relations. Par exemple, dans le domaine des assurances, considérer les contrats d'assurances comme des noeuds et leurs relations comme leurs degrés de similarité permet de trouver des groupes de contrats similaires, d'offrir des communautés à étudier pour en comprendre les relations, et de créer un modèle de score pour les nouveaux clients. Dans le réseau web, où les sommets sont des pages web et les arêtes des hyperliens, des groupes de pages ou de sites fortement connectés traitent souvent de thèmes apparentés et la détection de communautés permet l'amélioration des moteurs de recherche. En considérant le réseau de neurones du ver *Caenorhabditis elegans*, étudier l'organisation des liaisons et l'existence de structures communautaires peut permettre de comprendre si la formation des synapses entre neurones se fait de manière aléatoire ou non, ce qui peut avoir en biologie et en médecine de grandes répercussions, notamment en neurochirurgie. En marketing, grâce à un graphe de co-achats où les sommets représentent les produits et les arêtes le fait que les produits ont été achetés ensemble, un algorithme de détection de communautés permettra de trouver les profils des produits les plus co-achetés pour les analyser, en vue de proposer un système de recommandation (utilisé par exemple par la société *Amazon*).

La détection de communautés présente trois challenges :

Le premier est que les tailles des communautés peuvent être différentes au sein d'un même graphe selon la nature des objets étudiés. Par exemple, dans la section livres d'Amazon, les tailles des communautés seront plus grandes pour les livres dont le thème est fantastique que pour ceux dont le thème est historique. On peut observer ce phénomène en utilisant le site $http://www.yasiv.com/$.
Le second est que certains nœuds du graphe peuvent appartenir à plusieurs communautés. Par exemple, dans un réseau de collaboration scientifique, un auteur peut publier dans différents domaines et champs scientifiques.
Le troisième challenge concerne la taille des graphes. Si l'on considère de nouveau le graphe Amazon, qui possède plusieurs millions de nœuds et plusieurs centaines de millions d'arêtes, appliquer un algorithme de détection de communautés sur une seule machine peut dans le meilleur des cas prendre plusieurs heures, au pire, une erreur mémoire peut survenir due à la quantité de données traîtées.

0.1 Des réseaux complexes aux structures communautaires

Un graphe de terrain (ou réseau complexe) est un réseau constitué de données collectées correspondant à une vérité de terrain. Il est réalisé par des experts. Un graphe de terrain a des caractéristiques topologiques non triviales. Ces caractéristiques, qui n'apparaissent pas dans les graphes simples ou graphes aléatoires, émergent lors de la modélisation de systèmes réels. Ces réseaux mettent en exergue un ensemble d'individus, d'organisations ou d'objets reliés par des interactions sociales régulières ou incontestables. L'un des premiers ayant travaillé sur les relations humaines à travers les graphes fut Moreno (1934) qui publia "Who shall survive ?". Il étudie les affinités entre élèves de classes de divers degrés, représentés sous forme de sociogramme, posant ainsi les bases de la sociométrie. L'auteur explique par la suite l'importance de l'utilisation des graphes comme outils d'analyse sociologique, Moreno (1951).

C'est à Barnes (1954) que l'on doit le terme *analyse des réseaux sociaux*. Il porta son analyse sur un cas particulier de réseaux de terrains, *les réseaux sociologiques*. Wasserman et Faust (1994) écrivirent un livre sur les méthodes appliquées aux réseaux sociaux qui allaient ouvrir de très nombreuses portes comme celles du partitionnement de graphe et plus tard, la détection de communautés. Wasserman décrit un réseau social comme étant un ensemble fini d'acteurs avec des relations définies entre eux. La notion de graphes de terrains s'est construite au fur et à mesure d'analyses concernant des systèmes réels, en liaison avec les réseaux sociaux.

Les premières études utilisant les réseaux sociaux portèrent sur l'aspect relationnel entre personnes, Murdock (1949) et Lowie (1950). A travers un graphe social, l'objectif était d'expliquer les relations entre individus et comment les liens pou-

vaient se former. Elles suivent les études de Davis *et al.* (2009) sur l'aspect comportemental et relationnel entre individus en fonction de certains évènements. Les études portaient sur le fait de savoir si des personnes de même catégorie sociale allaient entrer en relation ou non. C'est en 1950 que les premières études portant sur les questions de groupes d'individus furent lancées par Homans (1950). Il exposa des exemples de groupes au sein de graphes sociaux. Ses travaux furent continués par Nadel (1957) qui montra l'existence de structures sociales intrinsèques aux réseaux réels. Les premières études des structures de groupes au sein du graphe furent proposées par Glanzer et Glaser (1961). Les méthodes étaient basées sur des graphes de communication. Les premiers travaux concernant la possibilité qu'un noeud d'un graphe social puisse appartenir à plusieurs groupes de noeuds furent introduits par Bonacich (1972) qui utilisa la topologique des graphes pour en exposer l'existence. Breiger (1974) proposa de dissocier les arêtes dans un groupe de noeuds très connectés de celles sortant de ces structures. L'idée fut d'expliquer les groupes de noeuds à travers les arêtes et de mettre une voix à la détection de telles structures. L'existence de structures de noeuds densément connectés entre eux et faiblement avec le reste du graphe fut exposée par Newman, qui employa le terme de *communautés* pour définir ces groupes de noeuds. Il montra que la présence de structures communautaires était une caractéristique des réseaux complexes. Des études menées notamment par Barabási et Bonabeau (2003), Newman (2003) et Clauset *et al.* (2009a) ont essayé de recenser les caractéristiques communes des réseaux complexes. Les caractéristiques portent sur l'effet "petit monde" (le fait que chaque individu puisse être relié à n'importe quel autre par une courte chaîne de relations sociales), un fort nombre de triangles entre groupes de noeuds fortement connectés et une distribution des degrés suivant une loi faible. Une étude portant sur les méthodes de détections de communautés fut proposée par Fortunato (2010). Il y référence plus d'une cinquantaine de méthodes (la liste n'est pas exhaustive) qu'il classifie selon leurs techniques de partitionnement. Ainsi, l'auteur montre les algorithmes spectraux, les méthodes hiérarchiques, les méthodes d'optimisation d'une fonction de qualité appelée la *modularité* et les autres méthodes, notamment celles concernant le recouvrement, où un noeud peut appartenir à plusieurs communautés. Le choix de cette classification n'est pas absolu. L'étude nous offre plusieurs conclusions à retenir. Tout d'abord, beaucoup de méthodes issues du partitionnement de graphes (découpage du graphe en groupes de noeuds de même taille dont le nombre est donné par l'utilisateur, chaque noeud appartenant à un et un seul groupe) avaient été appliquées ou transformées pour la détection de communautés. Parmi ces méthodes, citons celles portant sur l'optimisation d'une mesure de qualité qui est la modularité. Une étude menée par Brandes *et al.* (2008) montre que l'optimisation de la modularité est NP-difficile. Dès lors, les méthodes d'optimisation de la modularité ne peuvent s'effectuer qu'à une échelle locale si l'on veut pouvoir traiter des graphes de plusieurs millions de noeuds et d'arêtes. L'étude de Fortunato montre bien que les heuristiques les plus utilisées reposent sur les principes de classification hiérarchique, qui est subdivisée en deux catégories :

— les méthodes ascendantes (agglomératives) : partant des noeuds du graphe

(ensembles de singletons), où les algorithmes fusionnent deux classes à chaque itération en optimisant une certaine fonction de qualité.
— les méthodes descendantes (de subdivision), dans lesquelles on considère la topologie du graphe dans son entier pour y appliquer une coupe. A chaque itération, le graphe est scindé en deux, donnant deux sous-classes disjointes dont la coupe donne le score le plus élevé à une certaine fonction de qualité.

Ces deux familles de méthodes produisent une hiérarchie de communautés, que l'on appelle dendrogramme 6.1. Cependant, Fortunato (2010) a montré que ces méthodes n'avaient pas le même ordre de grandeur en termes de complexité. Les méthodes descendantes ont une complexité bien plus élevée que les méthodes ascendantes car elles considèrent le graphe dans son entier pour faire la coupe. La coupe optimale nécessite de tester plusieurs possibilités, parfois de migrer certains noeuds dans d'autres communautés comme le fait le recuit-simulé (Guimera et Amaral (2005)) maximisant la modularité, mais ne permettant l'application qu'à des graphes de quelques dizaines de milliers de noeuds en un temps raisonnable. Ces méthodes sont déterministes.

Les méthodes ascendantes, dont les calculs sont basés sur le voisinage des noeuds, demandent beaucoup moins de calculs, et permettent de traiter des graphes de plus grandes tailles. Les méthodes ascendantes sont non-déterministes alors que les méthodes descendantes le sont. Ces conclusions se retrouvent dans d'autres études comparatives telles que Plantié et Crampes (2013), Papadopoulos *et al.* (2012) ou Yang et Leskovec (2012). Les méthodes précédemment citées offrent la possibilité de partitionner un graphe en sous-groupes de noeuds disjoints, cependant, elles ne permettent pas de créer de recouvrements, où des noeuds peuvent appartenir à plusieurs communautés. C'est pourtant une caractéristique qui fut remarquée pour les graphes de terrain. Par exemple, en biologie, des recherches actuelles s'effectuent sur des réseaux d'interactions protéine-protéine pour permettre, entre autres, de prédire leurs fonctions. Cependant, des protéines peuvent posséder plusieurs fonctions, il est ainsi utile de construire des recouvrements, c'est à dire un système de classes chevauchantes, pour connaître les fonctions des protéines suivant les différents tissus.

Parmi les nombreuses méthodes de détection de communautés, la propagation de labels (Raghavan *et al.* (2007)) est celle qui est la plus rapide et celle permettant de traiter les plus grands graphes. Elle présente cependant certains inconvénients. C'est une méthode basée sur le vote majoritaire des voisins d'un noeud pour la propagation de labels, ainsi, en cas d'équidistribution des labels majoritaires, la méthode est sujette au non déterminisme. Il a été également noté que certaines mauvaises propagations pouvaient donner des communautés ayant le même label. Enfin, l'aspect asynchrone de la méthode, c'est à dire le fait que chaque noeud du graphe doive connaître le label courant de ses voisins en fait une méthode difficile à paralléliser. La propagation de labels constituera le socle algorithmique de nos contributions scientifiques.

0.2 Objectif de la thèse et plan du manuscrit

Les études comparatives menées par Fortunato ont montré l'existence de nombreuses méthodes dont les principales sont soit divisives, soit agglomératives. Il a été observé que les méthodes locales, c'est-à-dire dont le point de départ est atomique (par le noeud), permettaient de traîter de plus grands graphes que les méthodes divisives. La propagation de labels est une méthode locale, qui présente l'avantage d'être rapide et applicable à des graphes de plusieurs millions de noeuds et d'arêtes mais elle a certains inconvénients. C'est en ce sens que notre première contribution sera *i*) de proposer une version améliorée de la propagation de labels en y incluant une stabilisation par recherche de coeurs et mise en place de barrages artificiels pour éviter de mauvaises propagations. La seconde contribution *ii*) sera d'améliorer la méthode précédente pour le chevauchement en y incluant une fonction d'appartenance permettant de détecter des noeuds pouvant appartenir à plusieurs communautés. Plusieurs fonctions d'appartenance basées sur la densité et le coefficient de clustering seront proposées en vue d'une étude comparative. Enfin, nous proposerons *iii*) une implémentation MapReduce de notre propagation de labels pour travailler sur de plus grands graphes ayant au moins plusieurs millions de noeuds et d'arêtes. La propagation de labels dans sa forme asynchrone présente une difficulté pour le parallélisme à laquelle nous répondrons.

Dans la suite du manuscrit, nous présenterons au chapitre 1 la problématique de la détection de communautés disjointes et chevauchantes, les méthodes existantes pour la résolution de ces problèmes et les mesures sur la qualité de partitionnement qui y sont associées. Le chapitre 2 traitera des méthodes parallèles et distribuées de la littérature avec les plateformes destinées au traitement de grands graphes. Le chapitre 3 présentera notre contribution à la détection de communautés disjointes et chevauchantes.
Le chapitre 4 donnera une implémentation Hadoop parallèle et distribuée d'un de nos algorithmes pour la détection de communautés disjointes, ouvrant la voie à l'élaboration d'une implémentation pour le chevauchement. Enfin, nous présenterons nos conclusions au chapitre 5, nos travaux de recherches actuels et nos perspectives.

0.3 Graphes utilisés

Pour tester la qualité de nos algorithmes, et dans la mesure où nous voulons que nos algorithmes soient échelonnables selon l'ordre et la taille du graphe, le tout en répondant aux problématiques des grands graphes, nous proposons de travailler sur les réseaux réels, plus exactement sociaux.

Nous utilisons des graphes issus de la littérature dont nous connaissons par avance les communautés pour tester nos propositions algorithmiques et étudier leur comportement. Nous pouvons les retrouver sur le site
https : *//snap.stanford.edu/snap/*. Une étude comparative pourra ensuite être proposée avec les algorithmes les plus connus de la littérature. Les réseaux pour notre étude concernent :

Le club de karaté de Zachary, Zachary (1977) (Zac) : un réseau de membres d'un club chez lesquels une dispute entraine la formation de deux communautés autour du manager et de l'entraîneur.
Un réseau de dauphins, Lusseau *et al.* (2003) (Dol) : un graphe de dauphins étudié à Doubtful Sound, en Nouvelle Zélande. Le graphe comprend 62 dauphins représentés en deux communautés, mâles et femelles.
Un réseau footballistique, Girvan et Newman (2002a) (Foot) un graphe exposant 11 différentes compétitions entre clubs de football américain.
Un réseau de livres politiques américains, Krebs (2004) (pol) un graphe de co-achat exposant les livres politiques édités en 2004 et vendus sur *Amazon.com*. Le graphe comprend 3 communautés, les Démocrates, les Républicains et les neutres.
Un réseau de musiciens de jazz, Gleiser et Danon (2003), où chaque noeud représente un musicien, une arête dénotant que deux musiciens ont joué ensemble dans un groupe.
Le réseau aéroportuaire américain de 1997, Batagelj et Mrvar (2006), où les noeuds sont des aéroports et les liens sont des lignes directes .
Le réseau d'Amazon, Yang et Leskovec (2015), qui est un graphe de co-achat où les noeuds sont des produits (livres, vidéos, CD de musique,etc). Un lien est mis entre le produit i et le produit j si les deux produits sont fréquemment achetés ensemble.
Le réseau de collaborations scientifiques DBLP, Yang et Leskovec (2015), où les auteurs de publications scientifiques sont liés s'ils sont co-auteurs.
Le réseau de vidéos You tube, Yang et Leskovec (2015), où les noeuds sont des utilisateurs qui peuvent former des groupes auxquels s'agglomèrent de nouveaux utilisateurs. Les groupes de noeuds sont caractérisés par des thèmes sociaux.
Le réseau social LiveJournal, Yang et Leskovec (2015) est une communauté de blogs en ligne gratuite, où les utilisateurs se déclarent leur amitié, formant ainsi des liens. Les utilisateurs penvent également former des groupes sur des thèmes auxquels d'autres utilisateurs peuvent se joindre.
Les caractéristiques des graphes sont exposées sur le *tableau 1.*

Caractéristiques de certains réseaux réels				
réseaux	$\|V\|$ et $\|E\|$	densité	diamètre	AT
Zachary	34 \ 78	0,139	5	0,2557
Football	115 \ 615	0,0938	4	0,4072
Dauphins	62 \ 159	0,0841	8	0,3088
Politique	105 \ 441	0,0808	7	0,3484
Jazz	198 \ 5 484	0,281	6	0,52
US-Air 97	332 \ 2126	0.038	6	0.0
Netscience	1 589 \ 2 742	0,0021	17	0,6934
Amazon	334 863 \ 925 872	$1{,}65e-05$	44	0,3967
DBLP	317 080 \ 1 049 866	$2{,}08e-05$	21	0,6324
You tube	1 134 890 \ 2 987 624	$4{,}3e-06$	20	0,0808
LiveJournal	3 997 962 \ 34 681 189	$4{,}3e-06$	17	0,2843

Tableau 1 – Caractéristiques des réseaux utilisés fréquemment pour les études algorithmiques avec AT, pour la transitivité moyenne. $|V|$ est le nombre de noeuds du graphe et $|E|$, le nombre d'arêtes.

Chapitre 1

Etat de l'art

Sommaire

1.1 Introduction

La détection de communautés est un domaine de recherche actif depuis ces vingt dernières années. De très nombreuses approches ont été mises en œuvre pour la détection de structures communautaires. Certaines méthodes considèrent le graphe dans son ensemble et effectuent une coupe pour trouver des communautés alors que d'autres privilégieront une approche nodale (c'est-à-dire, un partitionnement fondé sur les propriétés de nœuds voisins). L'ère du traitement de données massives, a également vu la naissance d'architectures parallèles et distribuées sur lesquelles se sont agrégés de nombreux algorithmes et des solutions à des problèmes multidisciplinaires.

Ce chapitre vise à présenter les principales méthodes de détection de communautés disjointes et chevauchantes. Nous présenterons dans un premier temps la notion de réseaux complexes avec les caractéristiques communes qui leur sont associées. Nous donnerons une formulation à la détection de communautés disjointes et chevauchantes, ainsi que les méthodes respectives pour résoudre ces problèmes. A la fin de chaque section décrivant les méthodes pour un problème donné, un tableau récapitulatif suivi d'une discussion sera présenté pour expliquer les choix et les décisions que nous avons pris au cours de cette thèse. Nous exposerons les mesures supervisées et non supervisées relatives à chaque type de problème avec des tableaux récapitulatifs et une discussion sur le choix des mesures que nous avons sélectionnées. A partir de l'état de l'art que nous aurons présenté, nous tirerons les conclusions nécessaires pour le développement de nos solutions de détection de communautés disjointes, chevauchantes, parallèles et distribuées.

1.2 Notions et notations relatives à la théorie des graphes

Nous utilisons dans la suite de ce manuscrit de nombreuses notations qui illustreront l'état de l'art, nos propositions algorithmiques et nos expérimentations. Nous proposons dans cette section de donner les définitions générales relatives aux graphes.

Un graphe G est une structure mathématique, plus exactement un couple (V, E), où $V = \{v_1, ..., v_n\}$ est l'ensemble des sommets (ou nœuds) et E est l'ensemble des arêtes (ou liens) $E = \{e_1, ..., e_m\}$. Une arête $e_k \in E$ est un couple de sommets (v_i, v_j) reliant les sommets v_i à v_j. La notation usuelle pour définir un graphe G est $G = (V, E)$. Nous notons $|V| = n$ l'*ordre* du graphe, c'est-à-dire le nombre de sommets du graphe et $|E| = m$ le nombre d'arêtes. Deux sommets $u, v \in V$ sont dit voisins (ou adjacents) si une arête les relie, c'est-à-dire s'il existe une arête $e \in E$ telle que $e = \{u, v\}$. On dit également qu'une paire de sommets est *adjacente* si les deux sommets qui la composent sont adjacents.

Il existe des graphes orientés (ou dirigés) et non orientés (non dirigés). Un graphe orienté est un graphe pour lequel les arêtes sont orientées, ce qui n'est pas le cas pour un graphe non orienté. Un graphe peut également être pondéré ou valué, c'est-à-dire lorsqu'il existe une fonction $W : e \in E \to \mathbb{R}$ qui à chaque lien associe une valeur réelle. On note $G = (V, E, W)$ un graphe pondéré.

Un graphe a certaines caractéristiques propres. Nous donnons les principales définitions.

On note $N(u)$ le voisinage du nœud u , où $N(u) = \{u \in V, (u, v) \in E\}$. Le cardinal du voisinage d'un sommet est son *degré* (ou *degré d'incidence*) que l'on note $d(u)$ où k_u (Freeman (1978)) qui est le nombre de liens qui lui sont incidents.

Le degré moyen d'un graphe noté λ_G pour un graphe G est la moyenne des degrés des nœuds du graphe : $\lambda_G = \frac{1}{n} \sum_{u \in V} d(u)$.

Un graphe $G' = (V', E')$ est un *sous-graphe* de $G = (V, E)$ si $V' \subseteq V$ et $E' \subseteq E$, et toute arête de E' a ses extrémités dans V'. Etant donné un ensemble $V' \subseteq V$, le sous-graphe de G engendré (ou induit) par S' est le graphe $G' = (V', E')$, où $E' = \{(u, v) \in E : u, v \in S'\}$

Un *graphe partiel* de G est un graphe $G' = (V', E')$ qui a le même nombre de sommets mais pour lequel certains arcs ou arêtes ont été éliminés, c'est-à-dire avec $V' = V$ et $E' \subset E$.

On appelle *graphe complet*, un graphe où tous les sommets sont adjacents, c'est-à-dire si tout couple de sommets distincts est lié par une arête. Pour tout entier naturel n, on note K_n le graphe complet d'ordre n. Le nombre d'arêtes du graphe complet K_n est égal à $\frac{n(n-1)}{2}$. On appelle *clique* un sous-graphe complet de G.

Un chemin du sommet s vers le sommet t dans un graphe orienté est une suite $v_0, v_1, ..., v_k$ de sommets telle que $v_0 = s$, $v_k = t$, $(v_{i-1}, v_i) \in E$, pour tout $1 \leq i \leq k$. Le terme k est appelé la longueur du chemin, et on dit que le sommet t est joignable à partir du sommet s. Le chemin est dit simple (ou élémentaire) si les v_i sont distinctes deux-à-deux (arêtes incidentes deux-à-deux). La notion correspondante dans les graphes non orientés est celle de *chaîne*. Dans un graphe non orienté, un *cycle* est une suite d'arêtes consécutives (chaine) dont les deux sommets extrémités sont identiques, c'est-à-dire tel que $v_0 = v_k$.

La *distance géodésique* entre deux sommets dans un graphe est définie par la longueur d'un plus court chemin entre ces deux sommets.

Un graphe est dit *connexe* si deux sommets quelconques peuvent être reliés par un chemin.

La densité d'un graphe, noté ρ_G, est le rapport entre le nombre d'arêtes (ou d'arcs) et le nombre d'arêtes (ou d'arcs) possibles. $\rho_G = \frac{2m}{n(n-1)}$. Un graphe dont la densité est de 1 est un graphe complet, c'est-à-dire où chaque sommet est relié à tout autre sommet par une arête. Un graphe avec une densité nulle signifie qu'aucun sommet n'est connecté à un autre.

L'*excentricité* d'un sommet est sa distance maximale à tous les autres sommets. Le *diamètre* d'un graphe est la plus longue des distances entre deux sommets du graphe considéré. C'est l'excentricité maximale.

Il existe de très nombreux types de graphes dans la littérature. Nous nous focaliserons dans notre étude sur des graphes simples, c'est-à-dire ne contenant ni boucle (un lien reliant un sommet à lui même) et ni plus d'un lien entre deux mêmes sommets. Nous porterons notre attention aux réseaux complexes dont nous donnerons les caractéristiques générales dans cette section.

1.2.1 Représentation d'un graphe

Il existe plusieurs structures permettant de représenter un graphe. L'une des structures les plus intuitives concerne la représentation matricielle. Un graphe peut être représenté par une *matrice d'adjacence* de taille $|V|\times|V|$ dont l'élément non-diagonal noté A_{ij} représente le nombre d'arêtes liant le nœud i au nœud j. La matrice d'adjacence est notée A. Dans un graphe simple (sans boucle), la diagonale de la matrice ne comprend que des zéros. Cette représentation permet d'avoir des informations sur la topologie du graphe et sur les relations entre paires de sommets. On peut par exemple, connaître le nombre de chemins de longueur k entre deux nœuds i et j en élevant la matrice à la puissance k et en observant les éléments de la i^{eme} ligne et de la j^{eme} colonne de la matrice résultante.

Bien que la structure de matrice soit séduisante à utiliser, elle n'est pas pratique en programmation dans la mesure où elle demande un espace mémoire important même pour des machines modernes. Une représentation moins gourmande consiste à considérer une liste de voisins, nommée *liste d'adjacence*, notée $L_A = (l_{v_i})_{i=1}^{n}$ où l'élément l_{v_i} est la liste des voisins du sommet v_i.

1.2.2 Centralité dans un graphe

Il existe plusieurs mesures pour à la fois caractériser la topologie d'un graphe et révéler l'importance d'un nœud au sein du réseau. Nous proposons d'exposer les mesures les plus utilisées en analyse des réseaux sociaux.

Nous avons vu que le degré d'un nœud était le nombre de liens qui lui était incident. Cependant, cela ne donne aucune information sur son importance au

sein du graphe. L'une des premières mesures pour connaître l'importance d'un nœud dans un réseau est sa *centralité de degré normalisée*. Pour un nœud u, la centralité de degré normalisée consiste en $\bar{d}_u = \frac{d(u)}{n-1}$. Un nœud peut être connecté au plus à tous les autres nœuds du graphe, soit aux $n-1$ autres nœuds constituant le graphe. Ainsi, effectuer le rapport du degré du nœud u sur les $n-1$ autres nœuds permet de voir l'importance qu'a le nœud u vis-à-vis de son degré au sein du graphe. L'objectif également est de ramener la centralité dans l'intervalle $[0, 1]$. Pour un nœud u, plus sa centralité de degré normalisée est proche de 1, plus ce nœud sera connecté aux autres nœuds du graphes.

Cependant, il est intéressant d'examiner la variation de la centralité de degré des nœuds du graphe pour en étudier la distribution. En 1978, Freeman (1978) a ainsi proposé le *degré de centralisation du graphe* G comme étant $C_D = \frac{\sum_{i=1}^{n}[d(u*)-d(i)]}{[(n-1)(n-2)]}$ où $d(u*)$ est le degré du nœud maximal du réseau. Ce nombre varie entre 0 et 1. La valeur 1 est atteinte pour le graphe en forme d'étoile, c'est-à-dire un nœud connecté à tous les autres, tous de degré 1. La valeur 0 est atteinte avec une clique.

Il est également intéressant de comprendre l'organisation topologique du graphe et comment sont liés certains nœuds. Le cœfficient de clustering (CC) est une mesure d'analyse des réseaux sociaux et de regroupement des nœuds dans un réseau. Il mesure à quel point le voisinage d'un sommet est connecté, et calcule plus exactement la probabilité que deux nœuds liés à un autre nœud soient également liés. Le CC a une forme globale et une forme locale. La première (globale) concerne le graphe dans son ensemble alors que la seconde (locale) ne concerne que le nœud. Pour un nœud $u \in G$, le CC est défini par :

$$CC_u = \frac{\text{nombre de triangles contenant le nœud } u}{\text{nombre de triplets contenant le nœud } u} \tag{1.1}$$

où le triplet contenant le nœud u correspond au nombre de paires de voisins du sommet u. Par défaut, si le degré du nœud u est de 1 ou de 0, nous posons $CC_u = 0$. Le cœfficient de clustering global pour le graphe G est calculé en utilisant la valeur locale $CC_u, \forall u \in G$

$$CC(G) = \frac{1}{n} \sum_{u \in G} CC_u \tag{1.2}$$

Par définition, nous avons $0 \leq CC_u \leq 1, \forall u \in G$ et $0 \leq CC(G) \leq 1$. Pour un nœud u, plus grand est son coefficient de clustering, plus la probabilité que ses voisins soient liés est forte.

Il existe également des mesures pour connaître le degré avec lequel un nœud est directement connecté aux autres nœuds qui ne sont pas nécessairement directement connectés les uns avec les autres. La *centralité d'intermédiarité* (node betweenness centrality) correspond au nombre de plus courts chemins du graphe

passant par chaque sommet. Pour toutes paires de nœuds s et t d'un graphe G, la centralité d'intermédiarité est définie par :

$$CI(v) = \sum_{s \neq v, t \neq v, s \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{1.3}$$

avec σ_{st} le nombre de plus courts chemins entre s et t et $\sigma_{st}(v)$ le nombre de plus courts chemins entre s et t passant par v. Un niveau élevé de centralité d'intermédiarité n'est pas toujours corrélé avec un degré important du sommet. En effet, un nœud avec un faible degré faisant le lien entre deux groupes de sommets aura une centralité d'intermédiarité élevée. Dans sa conception, les nœuds qui ont une forte probabilité d'apparaître sur un court chemin choisi au hasard entre deux nœuds choisis également au hasard ont une haute intermédiarité.

Il existe également une *centralité d'intermédiarité pour les arêtes*. Soit $w : E \longmapsto \mathbb{R}$ la fonction de pondération sur les arêtes de G, avec E, ensemble d'arêtes du graphe G. Pour un graphe non pondéré, nous avons $w(e) = 1 \; \forall e \in E$. Soit un chemin entre deux sommets commençant en $s \in V$ et se terminant à $t \in V$. Notons σ_{st} le nombre de plus courts chemins entre les sommets s et t. La notion d'intermédiarité d'une arête repose sur le nombre de plus courts chemins qui passent à travers une certaine arête. La centralité d'intermédiarité pour les arêtes, que l'on note $CI(e)$ pour un lien e est donnée par :

$$CI(e) = \sum_{s,t \in V, s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}} \tag{1.4}$$

$\sigma_{st}(e)$ représentant le nombre de plus courts chemins allant de s à t passant par e. La complexité pour calculer la centralité d'intermédiarité est en $\mathcal{O}(n^3)$ (Brandes (2001)). Cependant, des recherches Brandes (2001), Bader *et al.* (2007) et Geisberger *et al.* (2008) portant sur l'approximation de cette mesure ont pu réduire la complexité en $\mathcal{O}(nm)$ sur des graphes complexes.

Il existe d'autres centralités en analyse des réseaux sociaux, nous invitons le lecteur à se reporter au glossaire du manuscrit.

1.2.3 Analyse des réseaux sociaux et graphes de terrains

La topologie des graphes dépend des systèmes complexes étudiés. Un réseau de collaboration scientifique n'aura pas exactement les mêmes caractéristiques qu'un réseau social lié à la musique ou au cinéma. Cependant, les graphes sociologiques ont des caractéristiques communes que nous allons développer. Dans notre étude, nous nous focaliserons sur des réseaux complexes (qualifiés de graphes de terrain). De nombreuses études, notamment celles de Barabási et Albert (1999), de Newman (2003) et de Clauset *et al.* (2009a) ont essayé de trouver toutes les caractéristiques liées aux réseaux complexes. Elles ont montré des caractéristiques communes concernant la distribution des degrés des nœuds,

un faible nombre de nœuds ayant une forte centralité (les *hubs*), un nombre de triangles important, une distance moyenne entre chaque paire de sommets assez faible et l'existence de groupes de nœuds fortement connectés ensemble et faiblement avec le reste du graphe, *les communautés*. Nous développons chacune des caractéristiques des réseaux complexes pour en venir aux structures communautaires et à la problématique de leur détection.

Distribution des degrés

De nombreuses études telles que Barabási et Albert (1999); Albert *et al.* (1999) et Clauset *et al.* (2009b) ont constaté que la majorité des graphes de terrain possèdent une distribution des degrés non uniforme, qui est approximée par une distribution en loi de puissance de type $P(k) = Ck^{-\gamma}$, avec $P(k)$ la proportion de nœuds de degré k et γ, appelé exposant d'invariance d'échelle, un réel strictement positif. Nous avons de manière générale $2 \leq \gamma \leq 3$. Ce type de graphe est qualifié de *réseau invariant d'échelle* (scale-free network). Lors de la création d'un graphe de terrain, les nœuds se connectent de manière non uniforme. Certains nœuds attirent les nouveaux nœuds en formant de nouvelles connexions. Ces nœuds que l'on pourrait qualifier d'attracteurs, caractérisés par une forte centralité, sont appelés *hubs* et sont typiques des graphes de terrains.

Effet petit-monde

L'effet du petit monde est une expérience menée en 1967 par les psychologues Travers et Milgram (1969) qui met en exergue l'hypothèse que la longueur de la chaîne des connaissances sociales requise pour lier une personne arbitrairement choisie à n'importe quelle autre sur terre est généralement courte. Le concept a engendré l'expression célèbre "le monde est petit (*It's a small world*)". Dans cette expérience, Stanley Milgram a mis en évidence des chaînes très courtes reliant deux citoyens aléatoirement choisis aux États-Unis (les chaînes obtenues avaient une longueur moyenne de six personnes, d'où l'expression qui en a découlé). En 2011, le site social Facebook publie une analyse de sa topologie et indique qu'il y a en moyenne cinq degrés de séparation entre ses membres (quatre, si l'on se réfère uniquement aux États-Unis). Ces expériences confirment qu'un petit nombre d'intermédiaires est suffisant pour connecter n'importe quelle personne à une autre en ce qui concerne les graphes sociologiques.

Variations de densité et de taille des structures communautaires et fort cœfficient de clustering

Dans un réseau social, chaque nœud est connecté à un certain nombre de sommets mais rarement à tous les sommets. Le degré moyen des graphes de terrains est très faible et indépendant du nombre de sommets du graphe (Albert *et al.* (1999)). Les graphes de terrain ont un cœfficient de clustering élevé.

C'est-à-dire que deux sommets voisins d'un nœud auront tendance à se connaître et donc à être liés. Ainsi, des nœuds dans certaines régions denses du graphe seront connectés à de nombreux triangles. On peut citer comme exemples des réseaux de collaborations avec un fort coefficient de clustering (Newman (2006)). Pour étudier les graphes de terrains, il faut donc considérer deux niveaux, celui concernant le nœud avec une densité élevée et celui de la topologie du graphe, de densité faible. Cependant, Melancon (2006) montre que la densité d'un graphe varie en fonction du domaine d'application en donnant des exemples réels. Ainsi, il n'existe pas de seuil universel permettant de dire si nous avons un graphe de terrain connaissant sa densité.

Structures communautaires

En 2002, Girvan et Newman (2002b) ont montré que la présence au sein de graphes sociaux de groupes de nœuds fortement connectés entre eux et faiblement avec le reste du graphe est une caractérisque des réseaux complexes, ils donnent le nom de communautés à ces groupes de nœuds fortement connectés. La difficulté de trouver des communautés est qu'elles peuvent avoir des ordres et des tailles différents au sein d'un même graphe. C'est en ce sens que nous allons dans la prochaine section formuler le problème général de détection de communautés et voir les différentes définitions des communautés (il n'existe pas de définition exacte).

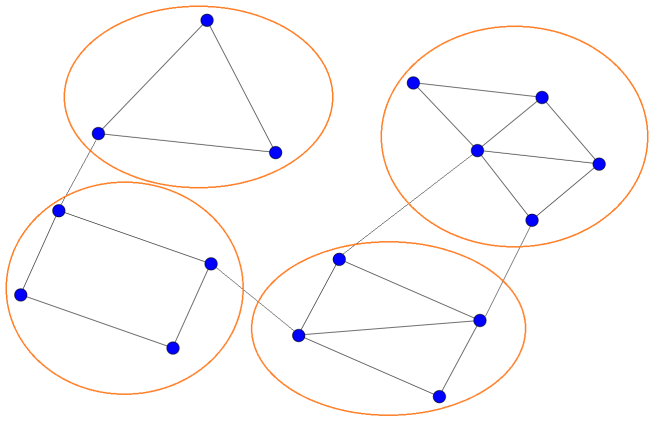


Figure 1.1 – Exemple d'un graphe avec quatre communautés

La modularité

La modularité est une mesure de qualité de partitionnement pour la détection de communauté créée par Newman et Girvan (2004). En considérant une partition P de l'ensemble des nœuds d'un graphe $G = (V, E)$, la modularité est une fonction prenant une partition $P = \{c_1, ..., c_k\}$ de communautés, $Q : P \rightarrow$

$[-1, 1]$, définie par $Q(P) = \frac{1}{2m}\sum_i (A_{ij} - \frac{k_i k_j}{2m})\delta(l_i, l_j)$, avec A_{ij} la matrice d'adjacence, k_i le degré du nœud i, m le nombre de liens dans le graphe, l_i l'identidiant de la communautés auquel appartient le nœud i, l_j l'identidiant de la communautés auquel appartient le nœud j, et $\delta(l_i, l_j) = 1$ si les nœuds i et j sont dans la même communauté, 0 sinon. Cette mesure est la somme, sur toutes les communautés des différences entre la proportion de liens à l'intérieur d'une communauté (ce qui équivaut à $\frac{A_{ij}}{2m}$ et la proportion de liens (soit $(\frac{k_i k_j}{2m})$ que devrait avoir une communauté dans un graphe aléatoire dont la distribution de degrés est la même que celui du graphe originel. Un tel graphe aléatoire est nommé le *modèle nul* (cf. glossaire 6.1). Une partition sera jugée correcte s'il y a plus de liens à l'intérieur de la communauté que ce à quoi l'on pourrait s'attendre. La modularité prend une valeur entre 1 et -1. Une valeur égale à -1 signifie qu'il n'existe aucun lien entre les nœuds d'une même communauté avec tous les liens pointant vers d'autres communautés. Une valeur de 1 signifie un grand nombre de liens dans les structures de communautés détectées. La valeur 0 de la modularité signifie la partition triviale considérant le graphe comme une grande communauté. L'inconvénient de la modularité est sa limite de résolution. En effet, si l'on est confronté à des communautés de tailles différentes à l'intérieur d'un même graphe, certaines communautés, même bien définies, pourront ne pas être distinguées dans la partition de modularité optimale. On ne pourra détecter des communautés ayant une taille inférieure à $\sqrt{m}$, m étant le nombre d'arêtes.

De nombreux algorithmes de détection de communautés que nous exposerons dans ce chapitre visent à maximiser cette métrique.

Notations scientifiques

Dans le cadre de ce travail doctoral, nous allons utiliser des notations scientifiques pour mieux expliquer certains algorithmes ou certaines mesures. C'est en ce sens que nous proposons le Tableau 1.1 de notation suivant :

1.3 Détection de communautés disjointes

Bien que la notion de communauté ne soit pas un objet mathématique encore défini, de nombreuses recherches ont essayé de trouver une formulation pour la définir. Nous allons voir que les définitions proposées par la littérature concernent des graphes spécifiques. Cette section vise à retracer les principales définitions de ce que l'on nomme communauté.

Sigle	Signification
$G = (V, E)$	Un graphe noté G, constitué d'un ensemble de sommets V et un ensemble d'arêtes E
$\|V\| = n$	Nombre de sommets
$\|E\| = m$	Nombre d'arêtes
k_{in}^S	Nombre de liens à l'intérieur du sous-graphe S
k_{out}^S	Nombre de liens sortant de S (liant les nœuds de S à $V - S$), c'est-à-dire ayant le nombre de liens qui ont une extrémité à l'intérieur de S et leurs autres extrémités à l'extérieur de S (soit $V - S$)
$N(u)$	Le voisinage d'un nœud u
$P = \{C_1, ..., C_r\}$	Une partition en r parties de sommets
$(u, v) \in E$	Un lien liant u et v
A	Matrice d'adjacence

Tableau 1.1 – Notations scientifiques

1.3.1 Formulation du problème de détections de communautés disjointes

Considérons un réseau social représenté par un graphe $G = (V, E)$. Le problème de détection de communauté dans sa forme générale consiste à trouver une partition $P = \{C_1, ..., C_r\}$ de l'ensemble des sommets V en r classes, avec $\bigcup_{k\in\{1,...,r\}} C_k = V$, $C_k \bigcap C_l = \emptyset$, $r \geq k \geq 1$ et $C_k \neq \emptyset, \forall k \in \{1, ..., r\}$, de telle sorte que les sommets dans une communauté soient fortement connectés et faiblement avec le reste du graphe.

Dans son livre sur la détection de communautés, Fortunato (2010) donne une formulation de la problématique de la détection de communautés fondée sur la mesure de Mancoridis *et al.* (1998), c'est-à-dire sur la densité intra-classe et inter-classe. C'est-à-dire qu'une communauté doit être caractérisée par une densité forte et un nombre de liens liant les communautés (inter-classe) faible. Ainsi, la problématique de la détection de communautés peut être vue comme une fonction à optimiser. En considérant un partition $C = \{C_1, ..., C_k\}$ en k parties de sommets disjoints, la mesure de Mancoridis se définit comme suit :

$$MQ = \frac{1}{k} \sum_i s(C_i, C_i) - \frac{1}{k(k-1) \sum_{i,j\neq i} s(C_i, C_j)} \tag{1.5}$$

avec $s(C_i, C_j) = \frac{|E(C_i,C_j)|}{|C_i||C_j|}$ avec $E(C_i, C_j)$ l'ensemble des liens étant à la fois dans la communauté C_i et C_j. En définissant par $\Delta_{Int}(C) = \frac{1}{k} \sum_i s(C_i, C_i)$ et $\Delta_{Ext}(C) = \frac{1}{k(k-1) \sum_{i,j\neq i} s(C_i,C_j)}$, nous obtenons $MQ = \Delta_{Int}(C) - \Delta_{Ext}(C)$. Dans cette dernière équation, $\Delta_{Int}(C)$ représente la cohésion interne des groupes

$C_1, ..., C_k$ (ou intra-classe) alors que $\Delta_{Ext}(C)$ représente la cohésion externe des groupes (ou inter-classe). Il s'agit de maximiser la somme de ratios sur l'ensemble des classes.

Le fait que les densités soient différentes d'un type de graphes à l'autre rend difficile la formulation d'une définition rigoureuse. En 2004, Radicchi *et al.* (2004) proposent de considérer deux types de communautés, les communautés au sens *faible* et les communautés au sens *fort*. En considérant un sous graphe G', auquel le nœud x appartient, le degré d'un nœud peut être coupé en deux parties, $k_x(G') = k_x^{in}(G') + k_x^{out}(G')$, où $k_x^{in}(G')$ représente le nombre d'arêtes reliant le nœud x à ses voisins dans G', et $k_x^{out}(G')$ représente le nombre d'arêtes sortant de G'. Ainsi, un sous-graphe G' sera considéré comme une communauté au sens fort si $k_x^{in}(G') > k_x^{out}(G'), \forall x \in G'$. Dans cette définition, chaque nœud a plus de connexions à l'intérieur de sa communauté qu'à l'extérieur. Une communauté au sens faible doit vérifier la condition suivante : $\sum_{x \in G'} k_x^{in}(G') > \sum_{x \in G'} k_x^{out}(G')$. Dans cette dernière définition, la somme de tous les degrés à l'intérieur de G' est plus grande que la somme de tous les degrés vers le reste du réseau.

Les critères dans l'article de Fortunato (2010) à la recherche de bonnes communautés s'articule autour de cinq grands axes :

- la réciprocité complète (les voisins de deux nœuds au sein d'une même communauté sont sensiblement les mêmes ((Luce et Perry (1949)))
- la joignabilité (deux nœuds d'une même communauté doivent pouvoir être proches topologiquement l'un de l'autre)
- le degré de sommet (les communautés doivent contenir des nœuds ayant un degré moyen important)
- un fort coefficient de clustering au sein des communautés (une communauté doit contenir des triangles et des triades)
- la comparaison de la *cohésion* interne et externe (nombre d'acteurs minimal qui pourrait rendre un graphe non connexe si on les retirait)

Le nombre de partitions possibles en k groupes de nœuds d'un graphe comprenant n sommets est *le nombre de Sterling de seconde espèce* $S(n, k)$ (Andrews (1976)). Le nombre total de partitions possibles est le n^{ieme} nombre de Bell, $B_n = \sum_{k=0}^{n} S(n, k)$ (Andrews (1976)). Lovász (1993) montre que pour n tendant vers l'infini, B_n a une forme asymptotique.

$$B_n \sim \frac{1}{\sqrt{n}} [\lambda(n)]^{n+1/2} e^{\lambda(n)-n-1} \tag{1.6}$$

où $\lambda(n) = e^{W(n)} = n/W(n)$, $W(n)$ étant la fonction de Lambert (Pólya et Szegö (1997)). Le problème de détection de communautés est donc un problème d'analyse combinatoire discret, où l'objectif est de trouver une partition avec des structures fortement denses et faiblement connectées avec le reste du graphe.

1.3.2 Approches divisives

Les approches divisives consistent à considérer la topologie entière du graphe et à effectuer une coupe pour obtenir un partitionnement. Une coupe d'un graphe est une partition des sommets en deux sous-ensembles. Les coupes ont lieu sur des liens connectant des régions denses du graphe. La méthode la plus connue est celle de la bissection (Fiedler (1973) et Pothen *et al.* (1990)) qui coupe le graphe en deux, puis opère de manière itérative sur les sous-graphes résultants. Il existe cependant d'autres méthodes que nous allons détailler dans cette section.

Approche spectrale

La méthode spectrale, issue de l'algèbre linéaire, établit notamment l'existence d'une base orthonormale de vecteurs propres pour tout endomorphisme symétrique sur un espace vectoriel complexe de dimension finie. Elle consiste en l'étude de matrices particulières, portant notamment sur les vecteurs propres de matrices définies positives. Une matrice définie positive est une matrice positive inversible et telle que pour tout vecteur $f \in \mathbb{R}$, avec f' la transposée de f, nous avons $f'Lf = \frac{1}{2}\sum_{i,j}^{n} w_{ij}(f_i - f_j)^2$. Cela induit certaines propriétés comme la positivité des valeurs propres, que l'on peut ordonner de la manière suivante $\lambda_1 = 0 \leq \lambda_2 \leq ... \leq \lambda_n$. On peut également par des méthodes algébriques ou numériques, calculer les valeurs et les vecteurs propres (Lanczos (1950)). Une étude théorique de la méthode spectrale a été traitée par Chung (1996) et Von Luxburg (2007).

Dans l'analyse spectrale des réseaux, la matrice Laplacienne a une place centrale :

$$L = D - A \tag{1.7}$$

où D est la matrice diagonale des degrés et A est la matrice d'adjacence.

Considérons la matrice Laplacienne L, les propriétés suivantes ont été établies (Von Luxburg (2007)) :

- L est une matrice symétrique et définie positive
- La plus petite valeur propre de L est 0, avec comme élément propre 1 (soit le graphe comme étant une communauté)
- Les éléments proposés de L sont réels et non négatifs $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$

La matrice Laplacienne permet également l'obtention d'information sur la topologie du graphe. Par exemple, l'ordre de multiplicité de la valeur propre 0 est égale au nombre de composantes connexes du graphe. Si G est un graphe connexe, la seconde plus petite valeur propre λ_2 est positive. Le vecteur propre associé à λ_2, appelée *vecteur de Fiedler*, fut étudié par Fiedler (1973) et Fiedler (1975). Une des applications est d'ordonner les sommets de G sur la droite des réels en associant chaque sommet v_i une composante x_i correspondante à sa

i^{ieme} composante du vecteur de Fiedler. En 1970, Hall (1970) a montré que si deux sommets v_i et v_j sont connectés, alors la distance $|x_i - x_j|$ dans le vecteur de Fiedler est petite. Ainsi, les sommets fortement connectés sont donc proches dans l'ordonnancement des sommets du vecteur de Fiedler. Il est ainsi possible d'effectuer une partition $P = \{P_1, P_2\}$ du graphe G en choisissant un réel r en posant $P_1 = \{v_i | x_i \leq r\}$ et $P_2 = \{v_i | x_i > r\}$. L'une des premières méthodes exploitant ce champ fut la méthode de Barnes et Hoffman (1981), dans laquelle la recherche considéra le vecteur de Fiedler et effectua une bissection entre les éléments positifs et négatifs du vecteur. Cette méthode permit de couper le graphe en deux. En 1990, Pothen *et al.* (1990) utilisèrent la médiane des éléments du vecteur de Fiedler pour la valeur de r pour effectuer une bissection récursive du graphe. De 1993 à 1995, Barnard et Simon (1994) proposèrent la bissection spectrale récursive et multi-niveaux. Cela consiste à effectuer une bissection en se fondant sur le vecteur de Fiedler et à réduire ce vecteur en minimisant la distance entre certains points. Le processus étant itératif, la méthode permit de couper le graphe de manière récursive. Une version parallèle fut développée par Barnard (1995). Mais ces deux méthodes n'utilisent qu'un seul vecteur propre, et donc une seule source d'information.
Shi et Malik (2000), ainsi Ng *et al.* (2001) eurent l'idée d'utiliser l'information stockée dans d'autres vecteurs propres pour améliorer la qualité de partitionnement. L'idée consistait à utiliser l'algorithme k-means dans l'espace propre afin de trouver k clusters et par transposition sur le graphe, k communautés. Les auteurs l'ont appliqué à différentes matrices Laplaciennes normalisées :

$$L_N = I - D^{\frac{1}{2}} A D^{-\frac{1}{2}} \tag{1.8}$$

et

$$L_N = I - A^{-1} D^{-\frac{1}{2}} \tag{1.9}$$

où I est la matrice d'identité de la matrice d'adjacence A. Les résultats en termes de qualité de partitionnement sont meilleurs que ceux de la matrice Laplacienne non normalisée. Malheureusement cette méthode nécessite de connaître la valeur k. Afin de remédier à ce problème, certaines recherches ont privilégié le nombre de vecteurs propres qui optimise le mieux une fonction de qualité de partitionnement pour la détection de communautés, comme la *modularité*.

Donetti et Muñoz (2004) proposèrent d'utiliser les vecteurs propres associés aux K plus petites valeurs propres non-nulles (au lieu simplement du vecteur de Fiedler), K étant un entier ne pouvant excéder le nombre de valeurs propres. L'idée est qu'un sommet est représenté dans un espace de dimension K constitué des composantes qui lui correspondent dans les K vecteurs propres. A chaque itération de l'algorithme, K est incrémenté de un et un algorithime de clustering hiérarchique est appliqué comme le single-linkage clustering ou complete-linkage clustering, fondé sur la distance angulaire entre les nœuds dans cet espace. Le meilleur partitionnement du dendrogramme est celui qui a la plus grande modularité. La configuration donnant la modularité maximale est alors conservée

pour retourner la partition en question parmi tous les dendrogrammes. La Figure 1.2 montre les résultats de l'algorithme appliqué à un graphe synthétique (Girvan et Newman (2002a)) de 128 nœuds, constitué de 4 communautés de 32 nœuds chacune, bien visibles.

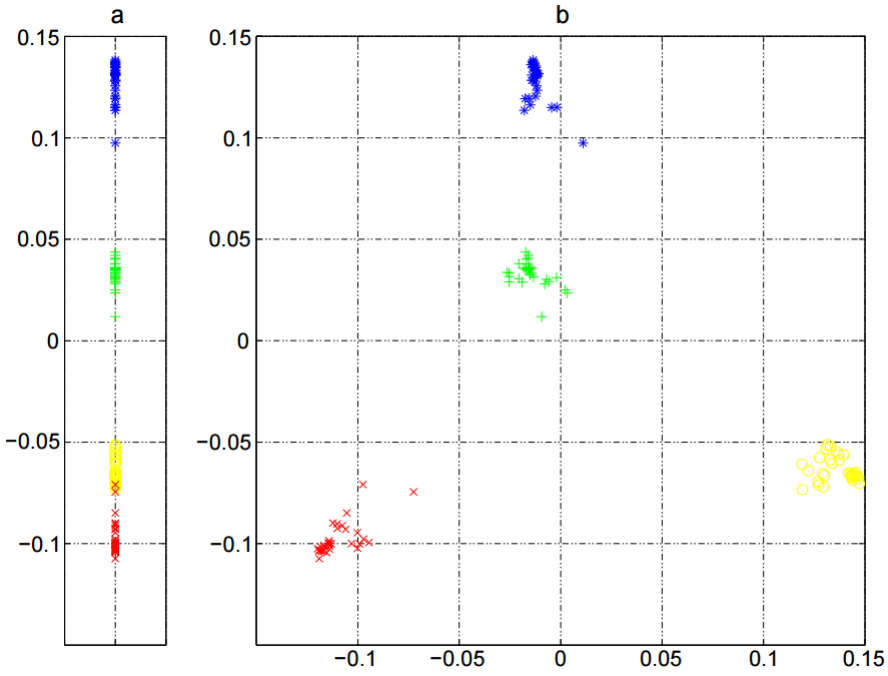


Figure 1.2 – a) Composante du premier vecteur propre non trivial (deux communautés sont clairement identifiées) b) Toutes les communautés peuvent être clairement identifiées lorsque les composantes du deuxième vecteur propre sont tracées par rapport à celles du premier (Extrait de Donetti et Muñoz (2004)).

Newman (2006) et Newman (2013) ont poussé l'étude en considérant la matrice de modularité pour y appliquer la méthode spectrale, tout en effectuant des méthodes de raffinage (où des nœuds changent dynamiquement de communautés).

Les méthodes pour calculer les vecteurs propres comme la méthode QR (Horn et Johnson (1985)) ou la méthode de Golub (1996) ont une très forte complexité, de l'ordre de $\mathcal{O}(n^3)$. Cela a pour conséquence que la méthode spectrale ne peut pas être appliquée à de grands graphes, tout au plus à des graphes de quelques milliers de nœuds. Des méthodes d'approximations existent cependant comme Koren *et al.* (2002) pour le calcul sur des matrices de taille relativement grande (leur plus grand graphe comprenant 7 533 224 sommets et 14 991 280 arêtes). La méthode spectrale est sujette à la propagation d'erreurs qui dérivent du calcul des composantes propres. A chaque calcul d'une composante propre,

une erreur de troncature est prise en compte pour le calcul du prochain vecteur propre. Parmi les matrices utilisées dans le domaine de la théorie spectrale, Newman (2006) a démontré que la modularité peut être exprimée en termes de valeurs propres et de vecteurs propres d'une matrice appelée *matrice de modularité*. En 2010, Shen et Cheng (2010) a effectué une étude comparative en utilisant différentes matrices utilisées en théorie spectrale.

La centralité d'intermédiarité

L'un des premiers algorithmes modernes pour la détection de communautés fut proposé par Girvan et Newman (2002b) avec la création d'une mesure d'importance fondée sur les arêtes, la centralité d'intermédiarité.

Girvan et Newman étendent le calcul de la centralité (pondéré) aux arêtes d'un graphe. Leur heuristique repose ensuite sur l'hypothèse que la suppression d'une arête de plus forte centralité est susceptible de rompre la connexité du graphe (ou plus exactement de la composante dans laquelle elle se trouve). On obtient une méthode hiérarchique divisive de clustering du graphe en supprimant tour à tour les arêtes de plus forte centralité (en prenant soin de mettre à jour la centralité après chaque suppression). Cependant, des recherches Brandes (2001); Bader *et al.* (2007); Geisberger *et al.* (2008) portant sur l'approximation de cette mesure ont pu réduire la complexité en $\mathcal{O}(n^2)$ sur des graphes complexes. Bien que ne pouvant pas être appliquée à de grands graphes, la méthode permet de classer les arêtes selon leur capacité à relier des communautés, ce qui a donné à cette méthode une forte notoriété.

Optimisation extrémale de la modularité

En 2005, Duch et Arenas (2005) proposent l'optimisation extrémale liée à la modularité. L'algorithme fonctionne à une échelle locale pour chaque nœud en optimisant la modularité. Initialement, une partition en deux groupes de nœuds est effectuée aléatoirement. Pour chaque nœud, une optimisation de la modularité est calculée en déplaçant le nœud considéré dans un des deux sous-groupes. Le déplacement d'un sommet d'un groupe à un autre a une incidence sur la modularité. On déplace, de manière répétée, les sommets entre les deux groupes jusqu'à trouver un découpage dont la modularité approche un maximum. Lorsqu'un état stable est établi, les arêtes liant les deux communautés sont retirées et le processus est répété de manière récursive jusqu'à ce que la modularité n'augmente plus.

La méthode présente une complexité algorithmique en $\mathcal{O}(n^2 log(n))$. Elle est cependant sujette au problème de résolution de limite (elle ne peut trouver de communautés dont la taille est inférieure à $\sqrt{m}$ (m étant le nombre de liens dans le graphe)). Les résultats sont encourageants mais le fait d'effectuer une bissection récursive ne permet pas de trouver toutes les structures communautaires. En 2006, Massen et Doye (2006) proposent deux modifications de la version du recuit-simulé précédent. La première modification consiste à

stopper l'algorithme de manière périodique, à essayer toutes les combinaisons de mouvements possibles et à prendre celle optimisant le plus la modularité. La deuxième amélioration consiste en l'utilisation de l'approche d'optimisation globale de Basin-Hopping (Wales et Doye (1997)) qui déplace des groupes de nœuds d'une communauté à une autre en optimisant toujours la modularité. Les résultats en termes de qualité sont meilleurs que ceux de Guimera *et al.* (2004), mais l'algorithme est plus lent. Ces algorithmes, utilisant toutes les combinaisons possibles, ne permettent pas de travailler sur de grands graphes.

1.3.3 Approches agglomératives et multi-niveaux

Une méthode héritée de celle du partitionnement est la méthode multi-niveau. Elle consiste à fusionner des nœuds pour produire de super-nœuds. A la fin du processus, les super-nœuds (qui agglomèrent des groupes de nœuds connectés) réprésentent les communautés.

En 2004, Clauset *et al.* (2004) ont proposé une méthode multi-niveau dont le processus de fusion se fait sur une optimisation globale de la modularité. C'est-à-dire que pour chaque paire de nœuds, toutes les fusions possibles entre nœuds voisins sont effectuées et l'on considère la fusion donnant la modularité maximale à travers tout le graphe pour une paire de nœuds voisins. Cette méthode donne des résultats en termes de qualité similaire à la méthode Newman et Girvan (2004). Les auteurs ont crée une structure de données adaptée afin d'améliorer la complexité de la méthode de Newman. Les auteurs ont en effet appliquées leurs méthodes sur un graphe de co-achat, celui d'Amazon, ayant 409 687 nœuds (représentant les items) et 2 464 630 arêtes. La méthode retourne de bons résultats en termes de partitionnement, avec une modularité de 0.745, qui révèle l'existence de structures communautaires.

En 2008, Blondel *et al.* (2008) fusionnent localement un nœud avec le voisin dont le résultat augmentera le plus une fonction de qualité, en l'occurence la modularité. Le processus (cf Algorithme 1) se poursuit de manière récursive sur le graphe résultant à chaque nouvelle fusion jusqu'à ce qu'il n'y ait plus d'augmentation de la modularité. Il s'agit de la version locale de la méthode de Clauset *et al.* (2004).

La méthode permet de produire des communautés de bonne qualité sur de petits réseaux et l'obtention de dendrogrammes. Elle ne souffre pas du problème fréquent que l'on peut trouver dans certains algorithmes optimisant la modularité, la résolution de limite. Cependant, pour de grands graphes (de plusieurs millions de nœuds et d'arêtes), l'optimisation d'une mesure globale conduit à une propagation d'erreur après la virgule. De ce fait, la qualité risque de se détériorer en fonction de la taille des réseaux. L'algorithme agissant localement, la méthode est instable, ne produisant jamais le même résultat d'un lancement

Algorithme 1 L'algorithme de Louvain

Entrée : Un graphe $G = (V, E)$

1: **A répéter jusqu'à l'obtention d'un score local optimal**
2: **Phase 1** : partitionner le réseau de manière gloutonne utilisant la modularité
3: **1)** Assigner à chaque nœud une communauté spécifique
4: **2)** Pour chaque nœud i du réseau
   — Pour chaque voisin j de i, choisir le voisin pour lequel l'assignation du nœud i dans une communauté augmenterait le plus la modularité
   — Répéter le processus jusqu'à ce qu'il n'y ait plus de changement
5: **Phase 2** : Agglomérer les sous-graphes en nouveaux nœuds
6: **1)** Laissons chaque communauté C_i former un nouveau nœud i
7: **2)** Laissons les arêtes entre les nouveaux nœuds i et j comme étant la réunion des arêtes entre les nœuds qui étaient dans C_i et C_j au sein du graphe précédent

à l'autre. Il a été montré que l'ordre jouait un rôle important sur la qualité des communautés détectées. La figure 1.3 montre un exemple de fonctionnement de la méthode de Louvain.

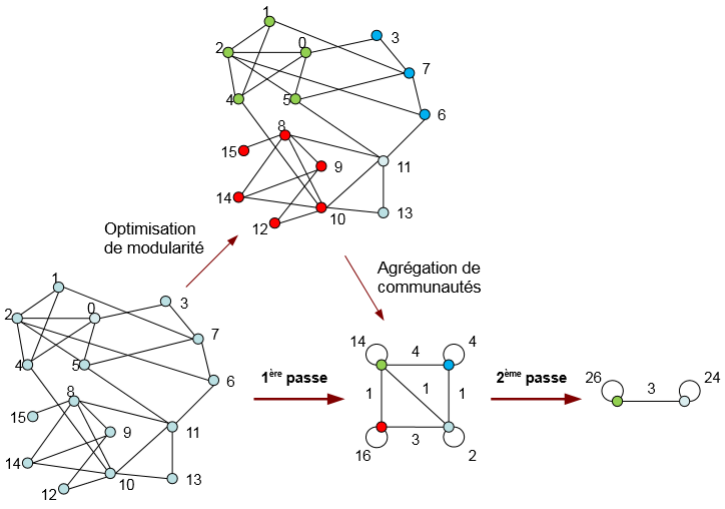


Figure 1.3 – Exemple d'application de la méthode de Louvain sur un graphe à 16 sommets (Extrait de Blondel *et al.* (2008))

La méthode de Louvain étant instable, une des pistes de recherche fut de la stabiliser. Une méthode proposée par Seifi *et al.* (2013) consiste à lancer plusieurs fois l'algorithme non déterministe et à considérer les nœuds qui apparaissent le plus souvent ensemble dans une même communauté. On appelle ces nœuds dont la fréquence d'apparition est très forte, des *cœurs*.

Définition Un cœur de communauté est l'ensemble des nœuds se trouvant fréquemment ensemble dans une même communauté après plusieurs lancements d'un algorithme non déterministe (exemple de Louvain avec Seifi *et al.* (2013)).

Cette méthode consiste à utiliser une matrice de fréquence, spécifiant le nombre de fois que chaque paire de nœuds apparaît dans les mêmes communautés. Les auteurs de cette méthode l'ont appliquée en utilisant la méthode de Louvain (Blondel *et al.* (2008)). Soit $\mathcal{N}$ le nombre de fois que l'algorithme non déterministe est lancé. A chaque essai, nous notons chaque paire de nœuds qui apparait dans une même communauté. Il est ainsi possible de définir une matrice $P_{ij}^{\mathcal{N}} = [p_{ij}]_{n\times n}^{\mathcal{N}}$ telle que p_{ij} représente la fréquence d'appartenance des nœuds i et j à une même communauté après les $\mathcal{N}$ essais. p_{ij}, étant une probabilité,$\forall(i,j) \in V \times V$ a une valeur comprise entre 0 et 1. Un nombre proche de 1 signifie que les nœuds i et$j$ sont souvent ensemble durant les $\mathcal{N}$ essais. Pour trouver les cœurs, on crée un nouveau graphe $G' = (V, E')$ où E' représente l'ensemble des arêtes créées à partir de la matrice de fréquence en utilisant un seuil $\alpha \in [0, 1]$ permettant de faire apparaître des composantes connexes. Le seuil α est un entier positif que l'on utilise pour créer le nouveau graphe à partir de la matrice de co-fréquence. Pour toutes paires de sommets du graphe, si la fréquence d'apparition dans la matrice est supérieure à la valeur α, on ajoute une arête au nouveau graphe G' reliant les nœuds en question. Par construction, les composantes connexes dans le graphe G' apparaissent. Les composantes connexes sont ici les cœurs, qui correspondent à nos communautés. G' est appelé le graphe α-seuillé. α est un paramètre qui influence des connexions dans le nouveau graphe G', et par conséquence, le nombre de composantes connexes. D'après les études menées par Seifi *et al.* (2013), de faibles valeurs de α conduisent à peu de composantes connexes alors que de fortes valeurs de α mènent à beaucoup de composantes connexes. Les auteurs ont montré qu'ils avaient réussi à stabiliser la méthode de Louvain, et à trouver des cœurs stables. La méthode présente l'avantage de pouvoir créer un dendrogramme en faisant varier α sur un intervalle.

Pons et Latapy (2006) ont proposé *WalkTrap* , un algorithme se fondant sur le fait qu'un marcheur aléatoire se promenant sur le graphe aura tendance à circuler dans des zones denses (ou à y rester plus longtemps) plutôt que dans des régions faiblement denses. Il s'agit d'un algorithme agglomératif permettant l'élaboration d'un dendrogramme. Considérons un nœud u et son voisinage $N(u)$. La probabilité que le marcheur aléatoire aille du nœud u vers un de ses voisins est de $\frac{1}{deg(u)}$. C'est ainsi que l'on peut calculer la probabilité que le marcheur aléatoire puisse, à partir de k pas, aller du nœud u à un autre nœud v du graphe. Les probabilités calculées entre chaque paire de nœuds vont être utilisées pour calculer les similarités entre chaque paire de sommets du graphe. WalkTrap démarre en considérant chaque nœud comme une communauté. L'algorithme calcule pour chaque nœud dans le graphe un vecteur qui donne la probabilité qu'un marcheur aléatoire arrive aux autres nœuds du réseau en k pas de temps. Ainsi, deux sommets u et v sont proches si leur vecteurs de proba-

bilité d'atteindre les autres sommets sont similaires. Les paires de nœuds ayant la plus grande similarité entre eux fusionnent pour former une nouvelle communauté. Un nouveau graphe est ainsi construit où les nœuds représentent des communautés. La méthode continue de manière récursive jusqu'à obtenir une communauté. A chaque découpage intermédiaire, un calcul de la modularité est effectué qui permettra de retourner la partition optimisant ce score. L'algorithme donne des résultats satisfaisants mais ne permet pas d'être appliqué à de très grands graphes dans la mesure où il faut calculer pour toutes paires de nœuds les distances. La complexité de l'algorithme est en $\mathcal{O}(n \times m \times H)$ où H est la hauteur du dendrogramme.

1.3.4 Approches fondées sur la détection de leaders

Dans un réseau, certains nœuds peuvent être plus importants que d'autres. Cela peut se traduire par une centralité ou un cœfficient de clustering plus fort pour certains nœuds du graphe par rapport à d'autres. C'est une des caractéristiques des graphes de terrains avec des nœuds situés au centre de structures communautaires et d'autres nœuds qui leur sont liés. Un graphe de terrain possède la caractéristique d'avoir une distribution des degrés des nœuds suivant une loi faible (Loi de Zipf) (Newman (2005)). Des algorithmes, ces dernières années, ont proposé de détecter les nœuds les plus importants que l'on peut qualifier de leaders et d'attribuer les autres nœuds à ces derniers pour former des communautés.

Shah et Zaman (2010) ont proposé l'algorithme "Leaders-Suiveurs". Les auteurs définissent deux types de nœuds, les *leaders* (nœuds qui connectent plusieurs communautés) et les *suiveurs loyaux* (nœuds dont le voisinage se situe dans une structure communautaire et possède un lien avec un leader). L'algorithme se décompose en deux étapes. La première étape consiste à rechercher les leaders. Les auteurs définissent une mesure de centralité, pour un nœud u, comme la somme des plus courtes distances de ce nœud à tous les autres nœuds du graphe. Un tri est alors effectué sur les nœuds, fondé sur cette mesure. L'ensemble des leaders est ainsi constitué de telle sorte qu'il ne puisse pas y avoir deux leaders voisins. La seconde étape consiste à assigner les autres nœuds qualifiés de suiveurs aux leaders. Les voisins directs des leaders leur sont assignés, permettant la création des communautés. L'algorithme est en $\mathcal{O}(n \times m)$. L'algorithme admet de meilleurs résultats en termes de qualité que les méthodes spectrales. Cependant, la méthode ne peut pas être appliquée à de très grands graphes dans la mesure où le calcul des plus courtes distances entre sommets nécessite un temps important. De plus, dans des zones fortement denses, l'algorithme peut détecter plusieurs leaders alors que cela ne devrait pas être le cas. La conséquence est la création de petites communautés qui auraient dû être fusionnées.

Kanawati (2011) a proposé LICOD (pour "Leader−driven algorithm for community detection in complex networks"). Il s'agit d'une version enrichie de la méthode présentée précédemment par Shah et Zaman. Les auteurs proposent différentes mesures pour le calcul des leaders, à savoir la centralité de degré (mesure locale) et la centralité d'intermédiarité des nœuds (mesure globale). Un nœud sera considéré comme leader si sa centralité est supérieure ou égale à $\sigma \in [0, 1]$ pourcentage de voisins. Deux leaders sont considérés comme étant dans une même communauté si leur nombre de voisins communs est supérieur à un seuil $\delta \in [0, 1]$. Concernant les nœuds suiveurs, les auteurs proposent d'utiliser l'inverse du plus court chemin vis-à-vis des communautés qui se constituent. Les résultats en termes de qualité sont corrects et surpassent les autres algorithmes sur des réseaux dont la densité est élevée. L'algorithme est cependant sujet au choix de la paramétrisation de σ et δ qui ont une influence sur le nombre de communautés trouvées. Comme pour la méthode de Shah et Zaman, l'algorithme est sujet au calcul des plus courts chemins dont le coût computationnel peut se révéler important pour de grands graphes. On notera que des travaux de recherches portant sur la centralité d'intermédiarité pour traiter des graphes relativement grands ont été effectués en utilisant le parallélisme par Kermarrec *et al.* (2011).

1.3.5 Approches fondées sur la perturbation du réseau

Les méthodes de perturbation des réseaux ont été créées pour l'amélioration d'algorithmes déterministes. L'idée est qu'en effectuant une modification topologique du graphe, certaines structures communautaires puissent être plus facilement détectées.

Gfeller *et al.* (2005) proposent de construire une suite de graphes G_1,G_2,...,G_n en modifiant la pondération des arêtes pour chaque couple de nœuds (x, y) via une loi de distribution uniforme. L'idée est que si des communautés existent au sein d'un graphe, une faible modification topologique du graphe en utilisant une nouvelle pondération des arêtes ne devrait pas modifier la structure des communautés détectées. Les auteurs définissent la probabilité intra-cluster p_{ij} comme étant le nombre de fois que les nœuds i et j se sont trouvés dans les mêmes communautés. Les liens ayant une pondération $p_{ij} \leq \theta$ sont retirés du graphe. Les composantes connexes résultantes forment ainsi les communautés. Bien que l'idée de perturbation des arêtes semble être un axe prometteur, l'évaluation des paramètres θ et σ handicape la méthode fortement. L'algorithme nécessite plusieurs tests avant de retourner sa meilleure partition.

Karrer *et al.* (2008) ont proposé un algorithme perturbatif qui enlève une certaine portion d'arêtes α et qui la remet entre certaines paires de sommets (x, y) avec une probabilité $\frac{d(x)d(y)}{2m}$, que l'on retrouve dans le modèle nul. L'objectif est de conserver la distribution des degrés des nœuds au sein du réseau.

L'idée de cette méthode est d'ajouter des liens dans des zones fortement denses et d'en retirer des zones faiblement denses. L'algorithme nécessite de tester cependant plusieurs valeurs de α avant l'obtention d'une partition de qualité. Cette méthode dépend également du modèle nul, qui parfois, ne met pas en évidence de structures communautaires ostensibles. L'algorithme présente cependant l'avantage de détecter des structures communautaires au sens fort et faible de Radicchi *et al.* (2004) et de tailles différentes.

Rosvall et Bergstrom (2007b) ont proposé *Infomod*. Il s'agit d'une méthode fondée sur la théorie de l'information, plus exactement sur la quantité d'information qu'une partition a vis-à-vis du graphe originel. L'idée consiste à partir d'une partition Y de sommets d'un graphe qu'un émetteur envoie à un receveur, de deviner la structure topologique du graphe X. La meilleure partition du signal Y contenant le plus d'information au sujet de X sera considérée. Chaque élément de Y représente une structure communautaire. L'information entre les deux partitions peut-être quantifiée par la minimisation de l'information conditionnelle de l'entropie $H(X|Y)$ de X sachant Y. $H(X|Y) = log[\prod_{i=1}^{q} \binom{\frac{n_i(n_i-1)}{2}}{l_{ii}} \prod_{i>j} \binom{n_i n_j}{l_{ij}}]$ où q est le nombre initialement choisi de communautés par l'utilisateur, n_i le nombre de sommets dans la communauté i et l_{ij} le nombre de liens entre les communautés i et j. A chaque calcul de la quantité d'information entre les deux partitions, des migrations de sommets s'effectuent entre communautés. La méthode donne des résultats en termes de qualité de partitionnement encourageants sur des graphes faiblement et fortement denses. Cependant, cette méthode ne peut pas être appliquée à des graphes de plus de 10^5 nœuds. La méthode nécessite également de donner un nombre de communautés initial. La figure 1.4 donne une explication du fonctionnement de la méthode Infomod.

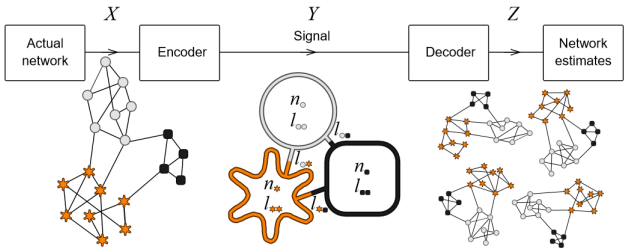


Figure 1.4 – Principe de base de la méthode Infomod. Un codeur envoie à un décodeur une information compressée sur la topologie du graphe à gauche. L'information donne une description grossière du graphe, qui est utilisée par le décodeur pour déduire la structure topologique originale. Extrait d'origine de Rosvall et Bergstrom (2007b)

Rosvall et Bergstrom (2010) ont proposé Infomap. Il s'agit d'une méthode fondée sur la théorie de l'information, un processus d'encodage et sur la marche aléatoire. Le procédé d'Infomap consiste à trouver des structures communautaires en étudiant la marche aléatoire et la ressemblance de motifs. Pour situer le marcheur aléatoire dans le graphe, on attribue aux sommets un identifiant

unique ainsi qu'un identifiant à la communauté à laquelle ils appartiennent. Pour décrire le déplacement du marcheur aléatoire, on commence par donner l'identifiant de la communauté, puis le label du nœud sur lequel il se trouve. Dès que le marcheur aléatoire sort de la communauté il faut lui attribuer l'identifiant de la nouvelle communauté et le label du nouveau nœud. L'idée est qu'il peut y avoir des structures ressemblantes dans chacune des communautés. Il y a par exemple, plusieurs "rue des Abeilles" en France, notamment à Montpellier, Mulhouse, Marseille, Toulouse ou à Lège-Cap-Ferret, dans des villes différentes. Les chemins qu'emprunte le marcheur aléatoire sont ainsi encodés et l'encodage minimum permet de trouver les structures communautaires.
Cette méthode fut testée sur les graphes aléatoires générés par les benchmarks de Lancichinetti *et al.* (2008) et surpassa les autres méthodes en termes de qualité de partitionnement, ce qui fonda sa notoriété. L'algorithme présente l'avantage de pouvoir traiter de très grands graphes. Il constitue à ce jour l'une des meilleures méthodes de partitionnement pour la détection de communautés disjointes.

De Meo *et al.* (2013) ont proposé CONCLUDE (pour Complex Network Cluster Detection). L'algorithme utilise à la fois l'importance des arêtes du graphe et un algorithme de clustering qui fait suite à une projection des nœuds dans un espace euclidien pour trouver les communautés. L'algorithme fonctionne en deux étapes. La première étape consiste en l'utilisation d'un modèle de propagation de l'information au sein du réseau, fondé sur une marche aléatoire avec non retour et de longueur fixe. Cela permet d'attribuer un score à l'importance des arêtes maintenant le graphe connecté. Les auteurs nomment cette mesure *la centralité d'arête* (Meo *et al.* (2011)). La centralité d'arête est ensuite utilisée pour projeter les sommets du réseau en points dans un espace euclidien et un calcul de distance entre chaque paire de nœuds est effectué. Un nouveau graphe est créé et la méthode de Louvain est appliquée.
Cette méthode nécessite cependant de paramétrer la longueur des marches aléatoires k qui est un paramètre global. Les graphes de terrain peuvent contenir des tailles de communautés différentes, ce qui pose un problème à la paramétrisation de k. L'algorithme a cependant une complexité proche de la linéarité en termes d'arêtes et produit, sous condition d'une bonne paramétrisation, de meilleurs résultats que la méthode de Louvain sur des graphes sociaux.

1.3.6 Approche par propagation de labels

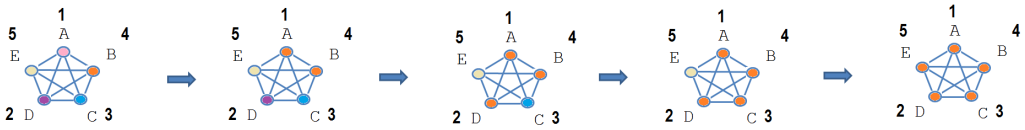
La méthode de propagation de labels (Raghavan *et al.* (2007)), notée LPA, est fondée sur la transmission d'un label d'un nœud à ses voisins. Un état d'équilibre est atteint lorsque chaque nœud a son label égal à celui de la majorité de ses voisins. Soit un graphe $G = (V, E)$, avec V l'ensemble des sommets ($|V| = n$) et E l'ensemble des arêtes ($|E| = m$).

A chaque étape, chaque nœud met à jour son label selon les labels de ses

voisins, en utilisant un vote. Le label du nœud u prendra le label majoritaire de ses voisins. En notant c_u le label du nœud u, et par $N^l(u)$ l'ensemble du voisinage du nœud u avec le label l, l'affectation d'un label au nœud u est donnée par la formule suivante :

$$c_u = \arg\max_l |N^l(u)| \tag{1.10}$$

A la fin du processus, les nœuds ayant le même label représentent une communauté. Cette méthode peut être effectuée de manière *synchrone* ou *asynchrone*. La méthode asynchrone signifie que la mise à jour d'un label d'un nœud est connue par tous les autres nœuds du graphe immédiatement. Son label est transmis pour la mise à jour des labels des autres nœuds. Ce n'est pas le cas du mode synchrone, où la mise à jour des labels utilise les labels des nœuds à la précédente propagation. La complexité de cet algorithme que cela soit en mode synchrone ou asynchrone est en $\mathcal{O}(k \times (n+m))$, où $k \in \mathbb{N}$ représente le nombre d'itérations de l'algorithme, spécifié par l'utilisateur. D'après les études menées par Raghavan *et al.* (2007), sur des graphes sociaux ayant moins de 1000 nœuds, 5 itérations suffisent pour obtenir une bonne classification. Pour des graphes ayant plus de nœuds, les recherches menées par l'auteur ne purent établir le nombre exact d'itérations. L'auteur préconise de mettre un nombre assez important d'itérations et d'observer itération après itération si le nombre de communautés évolue. Cet algorithme présente l'avantage d'avoir une complexité permettant de travailler sur de grands graphes.
Cependant, l'algorithme de propagation de labels présente l'inconvénient d'être instable, ne donnant que rarement le même résultat après plusieurs lancements. Il est aussi caractérisé par un problème intrinsèque conduisant dans certains cas à de très grandes communautés (monstres). Ce problème peut s'expliquer par deux raisons. La première est le choix parfois aléatoire du label que doit prendre un nœud lorsqu'il y a plusieurs labels majoritaires dans son voisinage. La seconde porte sur les tailles des structures. En effet, la propagation de labels prendra moins de temps à couvrir de petits réseaux que de gros réseaux. Pour pallier ces deux derniers problèmes, de nombreuses propositions algorithmiques ont été publiées depuis ces dernières années, que nous allons exposer.



$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ \mathtt{A} & \mathtt{D} & \mathtt{C} & \mathtt{B} & \mathtt{E} \end{pmatrix}$$

Figure 1.5 – Exemple de propagation de labels

Sur la Figure 1.5, nous considérons un graphe G avec $V = A, B, C, D, F$. Chaque nœud a initialement son propre label que nous avons modélisé par une couleur. Ainsi, le nœud A a la couleur rose, le nœud B a la couleur orange, le nœud C a la couleur bleue, le nœud D a la couleur violette et le nœud E a la couleur beige. Un ordre σ est donné sur les nœuds et est suivi pour procéder à la propagation de labels. Le nœud A a le choix entre les couleurs de B,C, D et E (cas de l'équidistribution des labels majoritaires), au hasard, le nœud A prend le label du nœud B, soit la couleur du nœud B, l'orange. La couleur majoritaire dans le voisinage du nœud D est orange, le nœud D prend la couleur orange. La couleur majoritaire dans le voisinage du nœud C est orange, le nœud C prend la couleur orange. La couleur majoritaire dans le voisinage du nœud B est orange, le nœud B prend la couleur orange. Enfin, la couleur majoritaire dans le voisinage du nœud E est orange, le nœud E prend la couleur orange. La première itération de labels est effectuée. Chaque nœud possède la couleur majoritaire de son voisinage, le processus s'arrête. Sur d'autres configurations de graphe, nous serions passés à la seconde itération de labels jusqu'à ce que chaque nœud possède la majorité des couleurs des voisins.

En appliquant ce procédé au graphe de la Figure 1.6, les communautés apparaissent.

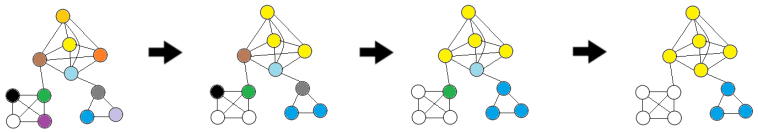
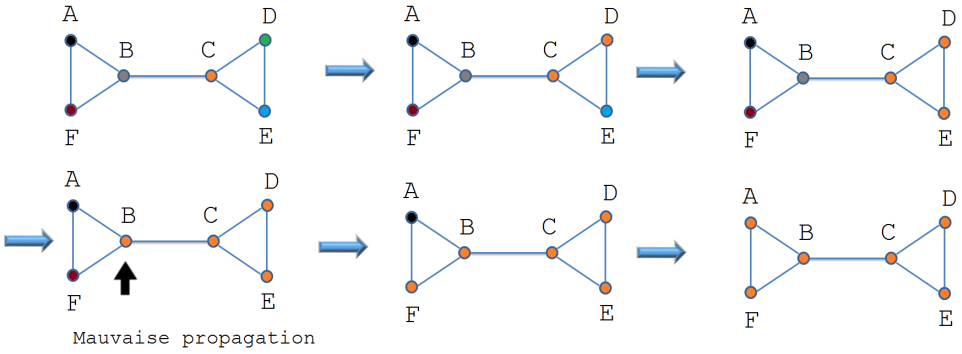


Figure 1.6 – Exemple de propagation de labels

Cependant, l'algorithme souffre de mauvaises propagations pouvant donner le phénomène de communautés géantes, c'est-à-dire l'incapacité pour l'algorithme de détecter de petites structures communautaires. Dans l'exemple de la Figure 1.7, une mauvaise propagation se fait lorsque le nœud B est visité pour la mise à jour de son label. A cause de l'équidistribution des couleurs majoritaires, le nœud B choisit au hasard une couleur, ce qui par la suite aura pour conséquence l'obtention de la communauté géante.

L'algorithme est non déterministe et souffre d'une forte instabilité. En considérant l'exemple du club de Karaté (Zachary (1977)), on peut s'apercevoir sur la Figure 1.8 que le nombre de communautés change au cours des différentes propagations de labels. Certains exemples donnent 2 ou 3 communautés alors que d'autres ne donnent qu'une grande communauté. Il s'agit d'un algorithme instable, c'est-à-dire ne donnant que rarement le même résultat d'un lancement à l'autre.

Leung *et al.* (2009) ont proposé un score pour chaque label qui diminue d'un certain pas $\delta \in [0, 1]$ quand la distance géodésique de la source du nœud qui a émis le label devient trop élevée. Cette méthode a été nommée "propagation de labels par atténuation". Cela a pour conséquence d'éviter l'obtention de trop grandes communautés dans les graphes. Les expériences ont montré que cette



$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ \mathrm{D} & \mathrm{E} & \mathrm{B} & \mathrm{A} & \mathrm{C} & \mathrm{F} \end{pmatrix}$$

Figure 1.7 – Exemple de propagation de labels donnant une communauté géante

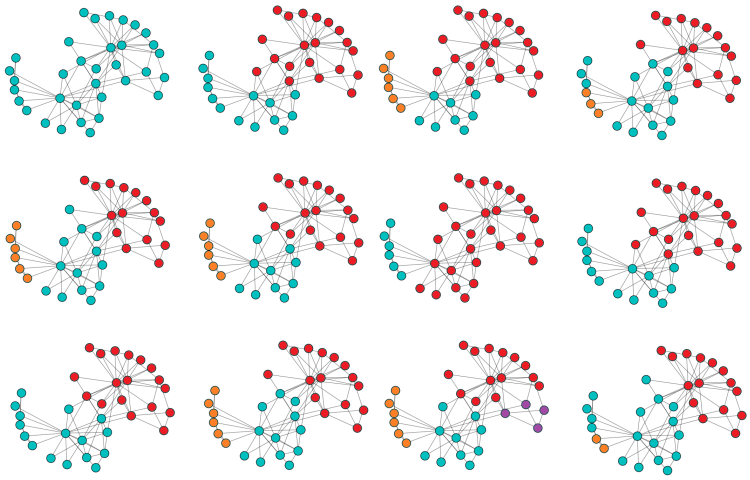


Figure 1.8 – La propagation de labels est un algorithme non déterministe et instable. Application sur le graphe de Karaté (Zachary (1977))

méthode permettait d'éviter de mauvaises propagations et donnait de meilleurs résultats que le LPA. Le problème restant est de pouvoir paramétrer la distance du nœud à la source afin de diminuer le score de la fonction d'atténuation. La solution n'est pas stable et nécessite plusieurs essais avant l'obtention d'un résultat concluant.

Šubelj et Bajec (2011) ont continué les travaux de Leung en proposant une propagation de labels offensive, défensive et hybride fondée sur la modifica-

tion du vote et du pas d'atténuation. Cette modification porte sur l'application d'une propagation de labels par préférence de nœud, qui consiste lors du vote, à prendre le label voisin dont le degré de centralité et la diffusion en termes de marche aléatoire sont les plus importants. Le facteur de préférence est la diffusion du label divisée par la centralité du nœud émetteur (méthode défensive pour la détection de cœurs), et la méthode offensive (permettant de définir les bordures des communautés) dont la distance de propagation est plus grande. Une hybridation des deux méthodes a mené à l'algorithme $K-$cœurs, qui consiste à appliquer la méthode défensive pour trouver les cœurs, puis à appliquer la méthode offensive pour trouver les frontières des communautés. Les résultats on montré de nettes différences entre les méthodes avec une stabilité plus forte pour la méthode défensive. La méthode défensive trouve de petites communautés alors que la méthode offensive trouve de très grandes communautés. Les résultats sont cependant équivalents au LPA en termes de qualité de partitionnement. Les expérimentations ont montré que la méthode hybride était un bon compromis dont le résultat en termes de qualité des communautés surpassait le LPA. L'algorithme nécessite cependant une paramétrisation importante, comme le pas d'atténuation et la puissance à laquelle on l'élève. L'algorithme n'est cependant pas déterministe.

Zong-Wen *et al.* (2014) proposent une propagation de labels guidée par consensus. Les auteurs proposent de lancer plusieurs fois l'algorithme de propagation de labels pour obtenir différentes partitions. Un nouveau graphe pondéré est alors créé où la pondération des arêtes représente le nombre de fois que chaque paire de nœuds est dans une même communauté. C'est alors qu'une propagation de labels avec consensus fondée à la fois sur les poids des arêtes et la fréquence des labels permet d'assigner à un nœud un nouveau label. Cette méthode produit de meilleurs résultats que le LPA mais une étude préalable pour chaque graphe doit être faite pour la paramétrisation. Cette méthode ne permet pas l'élaboration de dendrogramme.

Pour résoudre le problème de l'équidistribution des labels majoritaires pour le processus de vote, Xie et Szymanski (2013) proposent une propagation de labels avec opérateurs afin de stabiliser et de rendre déterministe le LPA tout en améliorant la qualité de partitionnement. Cela consiste à stocker, propager et trier les labels de chaque nœud en utilisant quatre opérateurs qui sont la *propagation*, *l'inflation*, *la coupure* et une phase de mise à jour de stabilisation du LPA. Chaque nœud reçoit le label majoritaire de ses voisins en éliminant les choix aléatoires (c'est-à-dire que s'il y a plusieurs labels majoritaires dans son voisinage, l'algorithme les ajoute à un vecteur et sélectionne le label recouvrant le groupe de nœuds ayant la centralité moyenne la plus élevée. Les communautés se forment itération après itération). La partition résultante est déterministe car il n'y a plus de choix aléatoires. Le *labelrank* a une complexité algorithmique en $\mathcal{O}(m)$.
Les auteurs ont proposé par la suite *LabelRankT* en y ajoutant une condition

supplémentaire sur la mise à jour des labels, destinée aux graphes dynamiques (graphes où les sommets et les arêtes sont ajoutés au cours du temps dans leur cas). L'idée est de considérer la partition actuelle pour inférer la partition suivante. Cela maintient l'ancienne partition tout en mettant à jour les nœuds seulement affectés par l'ajout d'arêtes. Lorsqu'une nouvelle arête se greffe dans le graphe, *LabelRankT* met à jour les nœuds qui sont sur ses extrémités. La qualité de partionnement est similaire à MCL et Informap sur des graphes sociaux, mais avec une complexité algorithmique moins élevée. Il est cependant plus rapide que certains algorithmes pour la détection de communautés dynamiques comme *facetNet* et *iLCD*. La complexité globale de cet algorithme entre deux images consécutives d'un même réseau est en $\mathcal{O}(k \times m)$, où k est le nombre d'itérations. *LabelRankT* peut être étendu à la détection de communautés chevauchantes en lui intégrant le SLPA ("*Speakerlistener Label Propagation Algorithm*") (Xie *et al.* (2011)) dont nous donnerons la définition en section 1.4.

Lou *et al.* (2013), arguent qu'une limitation du LPA est due à son processus de vote. En adoptant le label majoritaire de ses voisins l'information topologique est perdue et cela a pour effet l'instabilité du LPA et sa pauvre performance en termes de qualité dans certains cas. C'est en ce sens que les auteurs font intervenir une mesure nommée *"cohérence de proximité du voisinage"* (W-CNN) dont l'objectif est de calculer la probabilité qu'une paire de nœuds soit dans la même communauté. Elle est fondée sur un voisinage plus large des nœuds considérés lors du vote. Un label aura plus d'importance si sa source est issue d'un milieu fortement dense. Dans la phase de mise à jour des labels, un nœud adoptera le label ayant la plus forte valeur de W-CNN au lieu du vote majoritaire traditionnel.
Les auteurs montrent que pour de petits réseaux, les résultats sont équivalents au LPA sans toujours stabiliser la solution. L'algorithme toutefois montre de meilleurs résultats que le LPA pour de grands graphes de terrain, avec une meilleure stabilisation.

Xing *et al.* (2014) proposent une propagation de labels par nœuds influents. Cette méthode est fondée sur un ordre de mise à jour des labels et sur l'amélioration du vote lorsqu'il y a équidistribution des labels majoritaires. Cette méthode se fonde sur le (k)-shell d'un nœud et de son voisinage. Un k-shell est un sous graphe connexe maximum où chaque nœud a un degré au moins de k. Un k-shell est formé des sommets qui sont du k-core mais qui ne sont pas du $k+1$-core. La valeur du k-shell d'un nœud est k. Pour un nœud u, les auteurs notent $K_s(u)$ la valeur du k-shell du nœud u, $N(u)$ l'ensemble des voisins du nœud u et définissent *l'influence du nœud* par :

$$NI(u) = K_s(u) + \alpha \times \sum_{v \in N(u)} \frac{K_s(u)}{d(v)} \tag{1.11}$$

où $d(v)$, le degré du nœud u, $\alpha \in [0, 1]$ étant un paramètre global (utilisé pour

ajuster l'effet des voisins sur la centralité d'un nœud u). Les auteurs fixent l'ordre de mise à jour des labels de manière décroissante selon les scores des influences des nœuds. Ils définissent également *l'influence du label* fondée sur l'influence du nœud pour choisir un label lorsqu'il y a plusieurs labels majoritaires au sein d'un voisinage d'un nœud. De par leurs expérimentations, notamment sur des graphes réels comme le club de karaté, le réseau de dauphins, le réseau footballistique, celui des livres politiques et celui portant sur des collaborations scientifiques, les auteurs montrent qu'effectuer la mise à jour des labels des nœuds ayant un fort (k)-shell permet de mieux stabiliser la propagation de labels mais que les résultats en termes de qualité de partitionnement ne sont toujours satisfaisants. En effet, le NMI sur les dauphins n'est que de 0.65, celui du réseau footballistique n'est que de 0.87 et celui des livres politiques, de 0.656. Ce qui, par rapport à d'autres algorithmes notamment celui de Louvain montre que la méthode est moins performante. Les résultats sur les graphes aléatoires conduise a la même analyse.
Cette méthode est également sujette au choix de la valeur α qui reste un sujet de recherche ouvert. Par conséquent, cette méthode ne peut pas être appliquée sur des graphes réels sans faire une étude préalable. Une autre contrainte est le temps d'exécution nécessaire à la mise à jour de la fonction d'influence qui peut devenir très important pour de grands graphes.

Zhang *et al.* (2014) ont proposé une version modifiée du LPA avec la capacité à la prédiction de transition de percolation (LPAp). Les effets de la phase de prédiction au sein du LPA permettent de retarder la formation de communautés géantes. Les auteurs implémentent une condition de mise à jour dans le but de réduire le temps d'éxécution du LPAp.
Les expérimentations ont montré que LPAp permettait de trouver des communautés de petites tailles, était plus stable que le LPA et donnait de meilleurs résultats. Cependant, la mise à l'échelle pour de grands graphes semble être la faiblesse de la proposition algorithmique.

Cordasco et Gargano (2012) ont étudié le comportement de la propagation de labels synchrone et asynchrone. Dans le but de résoudre le problème d'une implémentation parallélisée de la propagation de labels (où la propagation se fait de manière séquentielle d'ordinaire), les auteurs proposent la propagation de labels semi-synchrone. Un algorithme de coloration est lancé permettant d'obtenir une partition $\mathcal{D} = \{D_1, D_2, ..., D_k\}$ (en k parties) de telle manière que deux nœuds adjacents n'aient pas la même couleur. Les couleurs obtenues serviront d'ordre de visite pour la mise à jour des labels. Tous les nœuds avec les mêmes couleurs effectueront en même temps la mise à jour de leurs labels selon le vote majoritaire classique du LPA. Durant cette mise à jour, les autres nœuds, dont la coloration est différente, sont bloqués, leur imposant l'impossibilité de mettre à jour leurs labels. Cela permet d'éviter de rendre l'algorithme synchrone et de perdre en qualité de partitionnement. Après que cette phase de mise à jour est effectuée, les nœuds qui étaient bloqués connaissent les nouveaux labels, no-

tamment s'il y a eu des modifications de labels dans leur voisinage. Une autre couleur est alors considérée et le processus continue. Lorsque toutes les couleurs ont été traîtées, une propagation de labels globale a été effectuée. On répète le processus un nombre de fois déterminé par l'utilisateur ou jusqu'à stabilisation en termes de vote.

Les auteurs ont montré que la méthode semi-synchrone était plus efficace en termes de stabilité que le LPA (sans la rendre pour autant déterministe), et parallélisable. Les résultats sont cependant équivalents au LPA en termes de qualité de partitionnement.

1.3.7 Autres méthodes

Méthodes de modularité extrémale

Guimera *et al.* (2004) proposèrent le recuit-simulé fondé sur l'optimisation de la modularité pour former les communautés. Une partition aléatoire est effectuée sur le réseau. Un nœud se déplacera dans une autre communauté si la modularité augmente ou avec une certaine probabilité. L'algorithme est itératif jusqu'à ce que la modularité n'augmente plus. Bien que l'algorithme donne des résultats satisfaisants, la méthode nécessite de paramétrer le recuit-simulé, et n'est pas exempte de tomber dans un optimum local duquel on ne puisse plus sortir. La complexité est difficile à estimer et dépend de la paramétrisation. De nombreuses combinaisons doivent être testées pour faire migrer un nœud d'une communauté à une autre, ce qui nécessite un certains de traitement. La méthode a été appliquée à des graphes ayant environ 200 nœuds.

Massen et Doye (2006) ont apporté deux améliorations majeures. La première est que l'algorithme s'arrête de manière périodique, évalue les communautés et teste toutes les possibilités de mouvement de certains nœuds qui optimisent le plus la modularité. La seconde réside en l'utilisation de l'approche Basin-Hopping (Wales et Doye (1997)), à savoir que des groupes de nœuds puissent changer de communauté d'un bloc. Cela permet d'éviter de tomber dans un optimum local concernant la modularité. L'algorithme donne de meilleurs résultats que celui de Guimera *et al.* (2004), mais est plus lent.

Modèle de Potts

Le modèle d'Ising (aussi appelé modèle de Lenz-Ising), est un modèle de physique statistique. Il représente un système de particules, chacune possédant deux niveaux d'énergie (on parle de *spins*). Le spin est le moment cinétique intrinsèque des particules quantiques. Il s'agit, pour simplifier, du sens de rotation de l'électron considéré. La spintronique (électronique de spin) permet de mettre

en évidence des phénomènes de transition de phase, c'est-à-dire une transformation du système étudié provoquée par la variation d'un paramètre extérieur particulier (température, champ magnétique...). En considèrant un système isolé de n particules auxquelles est associé un état (ou spin) -1 ou $+1$, on peut définir la *configuration du système*, qui est la donnée de chacun des spins du système. L'information peut donc être contenue dans un vecteur $\underline{x} \in \{+1, -1\}^n$ En notant par x_i la i^{eme} particule tel que $\underline{x} = \{x_i, 1 \leq i \leq n\}$, on définit la forme générale de l'hamiltonien comme étant $H(\underline{x}) = -\sum_{1 \leq i,j \leq n} J_{ij} x_i x_j$, où les J_{ij} sont les couplages entre les particules i et j (réels positifs ou nuls) et $x_i x_j$ est un produit scalaire. Suivant une certaine configuration du modèle, l'hamiltonien nous donne l'énergie du système de particules étudiées selon la configuration $\underline{x}$. L'hamiltonien est une fonction à optimiser. Plus l'énergie d'un système est élevée, moins le système est stable. Un système physique a tendance à se trouver dans un état d'énergie minimale. Le modèle de Potts consiste en une généralisation du modèle d'Ising où le système de spins peut être dans q états différents ($\underline{x} \in \{1, ..., q\}^n$).

C'est Fu et Anderson (1986) qui ont démontré par analogie qu'il existe une relation entre l'énergie des systèmes physiques (représentée par l'Hamiltonien) et la fonction de coût dans un problème d'optimisation discrète (problème combinatoire). Blatt *et al.* (1996) ont construit le modèle de clustering de Potts. C'est Reichardt et Bornholdt (2006) qui l'ont retranscrit au problème de la détection de communautés, avec l'énergie du système de spin (soit l'hamiltonien) équivalente à la fonction de qualité du regroupement à optimiser, les états de spins étant les indices communautaires. Le lien entre le modèle de Potts et la détection de communautés se fait en considérant les J_{ij} (couplages entre les particules i et j) comme étant la matrice d'adjacence A_{ij} (lien entre les nœuds i et j).

Reichardt et Bornholdt (2006) définissent l'hamiltonien comme étant :

$$H(\{\sigma\}) = -\sum_{i \neq j} a_{ij} A_{ij} \delta(\sigma_i, \sigma_j) + \sum_{i \neq j} b_{ij}(1 - A_{ij})(1 - \delta(\sigma_i, \sigma_j)) \tag{1.12}$$

$$+ \sum_{i \neq j} c_{ij} A_{ij}(1 - \delta(\sigma_i, \sigma_j)) - \sum_{i \neq j} d_{ij} A_{ij}(1 - \delta(\sigma_i, \sigma_j)) \tag{1.13}$$

où les A_{ij} sont les éléments de la matrice d'adjacence, $\sigma = \{\sigma_1, ..., \sigma_n\}$ est le vecteur représentant la partition du graphe, σ_i est le label de la communauté du nœud i et $a_{ij}, b_{ij}, c_{ij}, d_{ij}$ sont des pondérations sur les liens entre i et j. $\delta(\sigma_i, \sigma_j) = 1$ si les nœuds i et j sont dans la même communauté, 0 sinon. Jœrg Reichardt et Stefan Bornholdt ont voulu un algorithme permettant d'encourager la création de communautés à la fois avec une forte proportion de liens dans ces dernières et une faible proportion de liens en ressortant. Ainsi, peut-on expliquer la formule précédemment exposée comme suit :

1. $a_{ij} A_{ij} \delta(\sigma_i, \sigma_j)$ représente les liens internes
2. $b_{ij}(1 - A_{ij})(1 - \delta(\sigma_i, \sigma_j))$ représente les liens internes n'existant pas

3. $c_{ij}A_{ij}(1-\delta(\sigma_i,\sigma_j))$ représente les liens externes
4. $d_{ij}A_{ij}\delta(1-\sigma_i,\sigma_j)$ représente les liens externes n'existant pas

Pour les liens n'existant pas, le modèle de Jœrg Reichardt et Stefan Bornholdt utilise un graphe complet avec un poids (1 si le lien existe et 0 dans le cas contraire). Les auteurs ont réécrit la formule sous une autre forme :

$$H(\{\sigma\}) = -\sum_{i\neq j}(A_{ij}-\gamma p_{ij})\delta(\sigma_i,\sigma_j)) \tag{1.14}$$

où p_{ij} est la probabilité qu'il existe un lien entre les nœuds i et j dans le graphe nul (un graphe respectant la distribution des degrés des nœuds du réseau initial mais où les arêtes ont été mises de manière aléatoire). En considérant la dernière formule citée, supposons que le facteur γ n'apparaisse pas, nous obtenons alors la formule de la modularité. La modularité est une mesure de partitionnement souffrant d'une limite de résolution. Si des communautés sont de tailles différentes à l'intérieur d'un même graphe, certaines communautés, même bien définies, pourront ne pas être distinguées dans la partition de modularité optimale. Pour pallier ce problème, les auteurs font intervenir un facteur γ qui a une incidence sur l'échelonnabilité des tailles des communautés détectées. Pour $\gamma \to 0$, le graphe est considéré comme une seule communauté. Pour $\gamma \to \infty$, chaque nœud est considéré comme une communauté. On peut ainsi considérer la fonction hamiltonienne comme une fonction objective à optimiser, où les éléments à rechercher sont ceux du vecteur $\sigma = \{\sigma_1, ..., \sigma_n\}$. Les auteurs utilisent une méthode de recuit-simulé en partant d'un état initial où les spins sont assignés au hasard aux sommets, avec un nombre d'états q élevé. Les résultats expérimentaux en termes de qualité sont bons, mais cependant sujets au choix du paramêtre γ, à la lenteur de l'algorithme du recuit-simulé qui demande une forte paramétrisation et à sa complexité qui ne lui permet pas d'être appliqué à de très grands graphes.

Ronhovde et Nussinov (2010) ont dans des travaux postérieurs proposé des améliorations et diverses fonctions hamiltoniennes, mais également considéré d'autres algorithmes de détection de communautés dont les complexités sont moins fortes que celle du recuit-simulé. Les auteurs ont montré qu'en faisant intervenir γ, le problème de résolution de limite persistait dans des communautés de taille inférieure à $\sqrt{\gamma \times m}$. Ils introduisent le modèle absolu permettant de créer un modèle de Potts à q états sans utiliser le modèle nul, en supprimant par voie de conséquence le problème de résolution de limite. Pour ce faire, les auteurs proposent d'utiliser la densité comme probabilité et de maximiser son espérance sur toutes les communautés. L'algorithme utilisé est une méthode multi-niveau permettant la création d'un dendrogramme. Le système donne de meilleurs résultats que le modèle originel, mais nécessite deux grands paramètres, le nombre de contractions et le nombre d'états au départ (un nombre de communautés dont le nombre diminue au cours du temps). Les auteurs proposent par la suite d'autres fonctions hamiltoniennes comme pour des graphes

pondérés, un modèle fondé sur les graphes aléatoires d'Erdos-Renyi, et sa version pondérée. Les résultats en termes de qualité sont encourageants et très proches mais nécessitent la paramétrisation de γ qui a un impact très élevé sur la qualité des communautés.

Prat-Pérez *et al.* (2014) en 2014 ont proposé un algorithme de détection de communautés, SCD (pour Scalable Community Detection), fondé sur l'idée que des communautés doivent avoir une fort coefficient de clustering dans les réseaux complexes. SCD comprend deux étapes. La première consiste en l'obtention d'une partition P par optimisation du cœfficient de clustering. La deuxième étape consiste en un raffinage qui fait migrer certains nœuds dans d'autres communautés en utilisant un ordre sur les nœuds selon leurs coefficients de clustering. Les déplacements de nœuds s'effectuent jusqu'à stabilisation, c'est-à-dire à ce qu'il n'y ait plus de migrations. les auteurs proposent une estimation en $\mathcal{O}(m \times log(n))$. Cette dernière méthode d'estimation a été développée sur une architecture multi-cœurs permettant de travailler sur des graphes de plusieurs centaines de millions de nœuds ayant un milliard d'arêtes. Le temps de traitement est d'environ une heure avec une quarantaine de machines. Les machines ont des processeurs Intel Xeon E5530 de 2.4 Ghz, 32 GO de RAM et 1 térabit d'espace disque. Les résultats en termes de qualité de partitionnement sont satisfaisants, l'algorithme surpasse les autres sur sur DBLP, you Tube et live Journal avec des NMI respectivement de 0.17, 0.05 et 0.030, et des F_1-score respectivement de 0.38, 0.20 et 0.23. Walktrap ou Louvain donnent cependant de meilleurs résultats en termes de NMI sur Amazon et de DBLP avec des scores respectifs de 0.29 et 0.31.

Saltz *et al.* (2015) ont implémenté leurs méthodes sur Giraph (Hadoop) pour pouvoir considérer les très grands graphes. Ils ont testé leur méthode sur des graphes de plus d'une centaine de millions de nœuds et un milliard d'arêtes pour une durée d'une heure, en utilisant une quarantaine de machines.

1.3.8 Tableau récapitulatif des méthodes disjointes

Nous avons vu de nombreux algorithmes. Certains sont fondés sur la topologie globale du graphe pour effectuer une coupe alors que d'autres agissent de manière locale, vis-à-vis du nœud et de ses voisins. Certains autres sont hybrides, utilisant soit une méthode locale à laquelle sont adjointes des mesures sociales, soit en vis-à-vis d'un paramètre qui agrandit le voisinage d'un nœud à une certaine distance géodésique. Nous proposons un tableau récapitulalif, *Tableau 1.2*, permettant d'observer les caractéristiques globales des méthodes dans la mesure où nous souhaitons travailler sur de grands graphes de plusieurs millions d'arêtes.

Classification des principaux algorithmes de détection de communautés				
Algorithmes agglomératifs	Dend	complexité (ordre)	Par	Type
Blondel *et al.* (2008)	oui	$\mathcal{O}(m)$	MPI	local
Pons et Latapy (2006)	non	$\mathcal{O}(n^3)$	non	global
Ronhovde et Nussinov (2010)	non	$\mathcal{O}(m^\beta log(n))$ $\beta \sim 1.3$	non	local-hybride
Traag *et al.* (2013)	non	$\mathcal{O}(m^\beta log(n))$ $\beta \sim 1.3$	non	local-hybride
Newman (2004)	oui	$\mathcal{O}(nm^2)$	non	global
De Meo *et al.* (2013)	non	$\mathcal{O}(nm)$	non	hybride
divisifs				
Girvan et Newman (2002b)	non	$\mathcal{O}(nm^2)$	non	global
Donetti et Muñoz (2004)	non	$\mathcal{O}(mn^4)$	non	global
Newman (2006)	non	$\mathcal{O}(mn^3)$	non	global
Newman (2013)	non	$\mathcal{O}(mn^3)$	non	global
extrémaux				
Guimera *et al.* (2004)	non	$*$	non	global
LPA				
Raghavan *et al.* (2007)	non	$\mathcal{O}(n \times m)$	MPI Hadoop Spark	local
Leung *et al.* (2009)	non	$\mathcal{O}(n^2 \times m)$	non	local
Šubelj et Bajec (2011)	non	$\mathcal{O}(n^2 \times m)$	non	local
Lou *et al.* (2013)	non	$\mathcal{O}(n^2 \times m)$	non	local
Xing *et al.* (2014)	non	$\mathcal{O}(2n + 2km)$	non	
Cordasco et Gargano (2012)	non	$\mathcal{O}(n^2 \times m)$	non	hybride
Zhang *et al.* (2014)	non	$\mathcal{O}(n^2 \times m)$	non	hybride
Zhang *et al.* (2015)	non	$\mathcal{O}(n^2 \times m)$	non	hybride
Rezaei *et al.* (2015)	non	$\mathcal{O}(kn^2 + m)$	non	hybride
Shi et Zhang (2014a)	non	$\mathcal{O}(m + n)$	non	local
Shi et Zhang (2014b)	non	$\mathcal{O}(m)$	non	local
Hu *et al.* (2016)	non	$\mathcal{O}(km)$	non	hybride
Fang *et al.* (2016)	non	$\mathcal{O}(n^3 + nm)$	non	global
Peng *et al.* (2016)	non	$\mathcal{O}(n^2 \times m)$	non	hybride
Zhoua *et al.* (2016)	non	$\mathcal{O}(kn^2 + m)$	non	global
Liu *et al.* (2016)	non	$\mathcal{O}(kn + m)$	non	hybride
Théorie de l'information				
Rosvall et Bergstrom (2007b)	non	$*$	non	hybride
Rosvall et Bergstrom (2010)	non	$\mathcal{O}(m)$	non	local
Autres				
Radicchi *et al.* (2004)	non	$\mathcal{O}(\frac{m^4}{n^2})$	non	global
Clauset *et al.* (2004)	non	$\mathcal{O}(nlog(n)^2)$	oui	local
Duch et Arenas (2005)	non	$\mathcal{O}(n^2 log(n))$	non	local

Tableau 1.2 – Principaux algorithmes de détection de communautés. Le symbole "*" signifie que cela dépend du paramétrage, difficile à estimer. Le symbole "**", pour le type d'algorithme, signifie que cela dépend du paramétrage, un algorithme pouvant être local ou hybride. Les notations *Dend* et *Par* signifient respectivement "dendrogramme" et "parallélisé".

Nous pouvons observer, d'après le tableau 1.2, que les méthodes globales, qui utilisent la topologie du graphe dans son entier ont des complexités plus élevées que les méthodes locales, fondées sur les nœuds et leur voisinage. Nous

donnons des figures portant sur l'échelonnabilité pour observer les méthodes pouvant être appliquées à de grands graphes, avec des millions d'arêtes.

Nous avons recueilli, à travers les différents articles, la taille des graphes sur lesquels ont été appliqués les algorithmes que nous avons rencontrés. Nous observons, sur la figure 1.9 que les méthodes locales comme la propagation de labels ou la méthode de Louvain peuvent être appliquées à des graphes ayant une centaine de millions d'arêtes. Les méthodes divisives comme la méthode spectrale ou le recuit-simulé ayant une complexité élevée, ne peuvent être appliquées à de trop grands graphes. Les méthodes pour la détection de cœurs sont fondées sur le lancement en parallèle de méthodes locales, nécessitant une forte consommation mémoire et ne permettant pas l'application à des graphes de plus de cent mille arêtes.

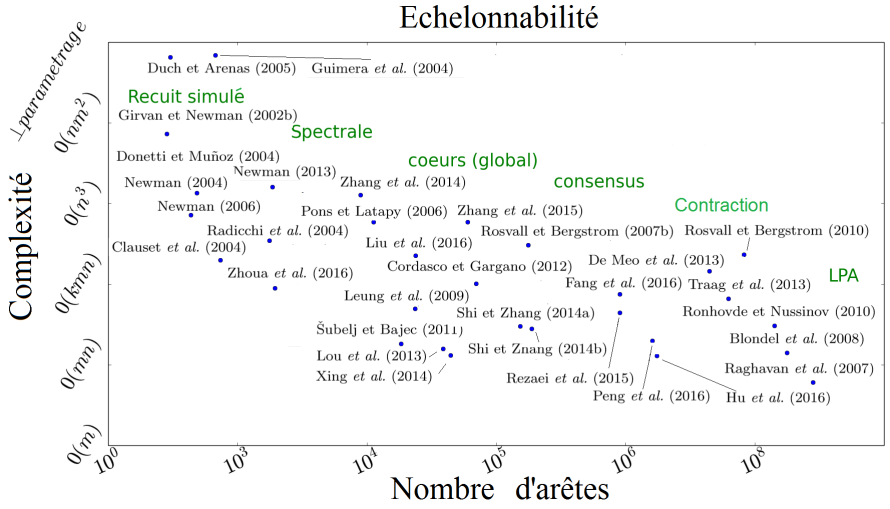


Figure 1.9 – Méthodes pour la détection de communautés disjointes.

1.3.9 Mesures supervisées et non supervisées pour la détection de communautés disjointes

Pour apprécier la qualité de partitionnement des algorithmes de détection de communautés, des mesures existent fondées sur la comparaison entre partition réelle (observée et évaluée par des experts) et partition trouvée par l'algorithme, il s'agit des mesures supervisées. Par exemple, si l'on considère un réseau de collaboration scientifique, un système expert va effectuer des regroupements entre auteurs ayant publié sur de mêmes thèmes scientifiques, cela donnera la vraie partition. Nous exposons les principales mesures, que nous utiliserons pour mener notre étude. Cependant, dans le cas où nous n'avons pas de connnaissance de la véritable partition, des mesures fondées sur le nombre de liens à l'intérieur des communautés et à l'extérieur, ou sur la comparaison de structures avec un graphe aléatoire sont utilisées, il s'agit dans ce cas des mesures non supervisées.

Mesures supervisées pour la détection de communautés disjointes

Les mesures supervisées permettent de comparer directement les résultats de l'algorithme proposé à la partition donnée par des experts. Dans certains, cas, elles peuvent également être utilisées pour observer la similarité des résultats entre plusieurs algorithmes. Nous présentons l'indice de Rand (et la version ajustée), l'information mutuelle normalisée et la pureté.

L'indice de Rand et la version ajustée (Rand (1971))
Considérons deux partitions P_1 et P_2 d'un même ensemble $S \subseteq V$. L'idée pour évaluer la ressemblance entre deux partitions consiste à mesurer le taux de bonnes assignations de ces paires d'observations. Pour chaque couple d'observations , on peut compter quatre types d'assignations possibles :

- N_{11} le nombre de paires de nœuds classées ensemble selon P_1 et P_2
- N_{10} le nombre de paires de nœuds classées ensemble selon P_1 et séparées selon P_2
- N_{01} le nombre de paires de nœuds séparées selon P_1 et classées ensemble selon P_2
- N_{00} le nombre de paires de nœuds séparées à la fois dans P_1 et dans P_2

L'indice de Rand est donné par la formule suivante :

$$Rand(P_1, P_2) = \frac{N_{11} + N_{00}}{N_{11} + N_{10} + N_{01} + N_{00}} \tag{1.15}$$

Cette mesure varie entre 0 et 1 et prend la valeur maximale en cas de parfaite correspondance. Cependant, cet indice ne demande pas d'établir un appariement entres les classes réelles et celles estimées, car seule est prise en compte la classification commune des différentes paires d'éléments dans les deux partitions. Cela induit deux problèmes majeurs. Premièrement, l'indice est fondé exclusivement sur les paires d'objets, et non les objets eux mêmes. Deuxièmement, une

inversion de classification pour deux éléments dans deux parties de petite taille sera moins pénalisée que si elle se produit dans des parties de plus grande taille. De plus, l'indice de Rand varie beaucoup entre deux partitions tirées au hasard. C'est pourquoi *l'indice de Rand ajusté* (ARI) (Hubert et Arabie (1985)) fut proposé dont l'espérance est nulle lorsque les partitions sont tirées aléatoirement. Il a pour forme :

$$ARI(P_1, P_2) = \frac{2(N_{11}N_{00} - N_{01}N_{10})}{(N_{01} + N_{00})(N_{11} + N_{01}) + (N_{00} + N_{10})(N_{10} + N_{11})} \quad (1.16)$$

Sa valeur est comprise entre 0 et 1. Elle prend la valeur 1 lorsque les deux partitions sont identiques. Si la valeur est proche de 0, les deux partitions sont très différentes.

La pureté (Manning *et al.* (2008)) : Cette mesure, pour une communauté $c_i \in P_1$, avec $P_1 = \{c_1, ..., c_m\}$, par rapport à une autre partition P_2 (avec $P_2 = \{c'_1, ..., c'_n\}$) se définit comme suit :

$$purete(c_i, P_2) = \max_{1 \leq j \leq n} \frac{|c_i \cap c'_j|}{|c_i|} \quad (1.17)$$

Cette fonction calcule le taux de recouvrement maximal entre la communauté c_i et les communautés se trouvant dans la partition P_2. On définit la pureté de la partition P_1 par rapport à la partition P_2 par la somme pondérée de la pureté des communautés de P_1 par rapport à P_2, ce qui nous donne :

$$purete(P_1, P_2) = \sum_{i=1}^{m} w_{c_i} \times purete(c_i, P_2) \quad (1.18)$$

où $w_{c_i} = \frac{|c_i|}{\sum_{i=1}^{m} |c_i|}$

L'information mutuelle normalisée (NMI) (Kullback (1959)) : C'est une mesure qui permet de représenter le degré de dépendance entre deux partitions P_1 et P_2. Elle est fondée sur la théorie de l'information. La probabilité qu'un nœud choisi au hasard dans une partition P_1 appartienne à la communauté k est $P(k) = \frac{n_k}{n}$ où n_k est le nombre de nœuds dans la communauté k et n est le nombre total de nœuds du système. L'entropie de Shannon, en utilisant la distance de Kullback-Leibler, est définie comme : $H(P_1) = -\sum_{k=1}^{|P_1|} \frac{n_k}{n} log_2 \frac{n_k}{n}$, où $|P_1|$ est le nombre de communautés dans la partition P_1. L'entropie de Shannon de la partition P_1 représente la quantité d'information fournie par cette même partition sur la topologie du graphe en termes de communautés.

L'information mutuelle $I(P_1, P_2)$ évalue le niveau d'inter-dépendance entre deux partitions d'un graphe. Il est possible de définir une matrice de confusion pour les partitions P_1 et P_2 en identifiant combien de nœuds n_{ij} de la communauté i de la partition P_1 sont dans la communauté j de la partition P_2. L'information mutuelle est $I(P_1, P_2) = \sum_{i=1}^{P_1} \sum_{j=1}^{P_2} \frac{n_{ij}}{n} log_2(\frac{n_{ij}}{n_i n_j})$ où n_i est le

nombre de nœuds de la communauté i de la partition P_1 et n_j est le nombre de nœuds de la communauté j de la partition P_2. La variation de l'information $V(P_1, P_2)$ est $V(P_1, P_2) = H(P_1) + H(P_2) - 2I(P_1, P_2)$ qui mesure la "distance" de l'information entre les deux partitions P_1 et P_2. Pour l'obtention d'une valeur comprise entre 0 et 1, on définit l'information mutuelle normalisée comme étant $NMI(P_1, P_2) = \frac{I(P_1,P_2)}{\sqrt{H(P_1)+H(P_2)}}$. Sa valeur peut varier entre 0 et 1. Plus la valeur est proche de 1, plus les deux partitions sont identiques.

Le score F_1 (van Rijsbergen (1979)), couramment utilisé dans la recherche d'information, peut être vu comme une mesure d'exactitude de la précision et du rappel entre deux partitions. En considérant deux communautés A et B, le score F_1 est définit par :

$$F_1(A, B) = 2\frac{|A \cap B|}{|A| + |B|} \tag{1.19}$$

Plus les communautés A et B partagent de nœuds en commun, plus le score F_1 est proche de 1. Un score nul signifie que les deux communautés ne partagent aucun nœud en commun. Cette méthode est fondé sur la combinaison de la précision ($precision(A, B) = \frac{|A\cap B|}{|A|}$) et du rappel ($rappel(A, B) = \frac{|A\cap B|}{|B|}$).

Considérons la partition trouvée par l'algorithme de détection de communautés $P_1 = \{c_1, c_2, ..., c_t\}$ et la partition représentant l'ensemble des communautés de vérité de terrains $P_2 = \{c'_1, ..., c'_k\}$. On note que le nombre de communautés peut être différent entre les deux partitions. La moyenne du score F_1 que nous utilisons est celle concernant les communautés trouvées par un algorithme où nous comparons chacune d'entres elles avec la communauté la plus similaire parmi les communautés de la partition de vérité de terrain, c'est-à-dire, avec la communauté de vérité de terrain partageant le plus de nœuds en commun. Nous avons ainsi :

$$\bar{F_1}(P_1, P_2) = \frac{1}{t}\sum_{i=1}^{t} \max_{j\in[\![1,k]\!]} F_1(c_i, c'_i) \tag{1.20}$$

Cette valeur illustre à quel point chaque communauté que l'algorithme a trouvée est similaire aux communautés de la partition de vérité de terrain, en regardant les communautées entre les deux partitions partageant le plus de nœuds en commun. Une valeur proche de 0 signifie que les deux partitions sont totalement différentes, sans aucune similarité entre communautés détectées et réelles alors qu'une valeur proche de 1 signifie que les deux partitions sont très semblables avec les mêmes nœuds à l'intérieur des mêmes communautés.

Mesures non supervisées pour la détection de communautés disjointes

Les mesures non supervisées permettent d'avoir une estimation de la qualité de partitionnement sans connaître les véritables partitions. Ces mesures sont soit fondées sur des structures aléatoires respectant la topologie du graphe initial, soit fondées sur les densités et les liens entre communautés. Nous utilisons

dans nos futures études la modularité et la conductance.

La conductance (Kannan *et al.* (2004)) : Cette mesure est fondée sur la densité des communautés et le nombre de liens sortant de celles-ci. Une structure communautaire est supposée avoir beaucoup de liens en son sein et un nombre faible de liens sortants. La conductance est fondée sur le rapport entre le nombre de liens sortants, l^c_{out}, et le nombre total de liens (intérieurs l^c_{int}) pour une communauté c.

En considérant une communauté c d'un graphe G, avec $c = (V_c, E_c)$ (V_c l'ensemble des sommets de c et E_c l'ensemble des arêtes de c), la conductance de cette communauté est définie par $\varphi_{(c,G)} = \frac{l^c_{out}}{[2l^c_{int}+l^c_{out}]}$. En considérant une partition $P = \{c_1, ..., c_k\}$ en k parties de nœuds disjoints, la conductance de G se définit comme suit :

$$\Phi_G = \frac{1}{k}[\sum_{c=1}^{k} \varphi_{(c,G)}] \tag{1.21}$$

$$= \frac{1}{k}[\sum_{c=1}^{k} \frac{l^c_{out}}{[2l^c_{int} + l^c_{out}]}] \tag{1.22}$$

La conductance peut avoir une valeur comprise entre 0 et 1. Plus cette valeur sera proche de 0, plus cela signifiera que les communautés ont une densité forte avec peu de liens pointant vers l'extérieur.

La conductance ne peut être utilisée que dans certains cas précis. On considère dans un premier temps que la communauté c sur laquelle nous appliquons la conductance est connexe, plusieurs cas sont à étudier :

- Si $l^c_{int} \gg l^c_{out}$, la conductance sera proche de 0. Le nombre de liens sortant de c (ayant une extrémité dans c et une autre dans $V - V_c$) est très faible par rapport au nombre de liens à l'intérieur de c. Cela induit, par l'hypothèse de la connexité de c, que c a une forte densité et cela sera d'autant plus vrai que c sera proche d'une clique. Ces conditions induisent la présence d'une structure communautaire.
- Si $l^c_{out} \gg l^c_{int}$, la conductance $\varphi_{(c,G)}$ sera proche de 1. Cela entraîne que le nombre de liens sortant de c sera beaucoup plus important que le nombre de liens à l'intérieur de c, ce qui n'est pas typique d'une structure communautaire.

Si la communauté c n'est pas connexe, c'est-à-dire qu'il y a des nœuds isolés dans c, alors la formule ne peut être utilisée comme mesure de qualité pour la détection de communautés car le nombre de nœuds isolés (dont le degré est nul) n'a aucun impact sur la conductance, en effet, les nœuds isolés n'agissent pas sur le nombre de liens. Ainsi, si une communauté c n'est pas reliée au reste du graphe et qu'il existe des nœuds isolés en son sein, nous aurons une conductance faible sans pour autant avoir détecté une structure communautaire.

Nous illustrons les cas avec des exemples simples. Considérons un graphe $G = (V, E)$ où $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12\}$, décomposé en deux communautés $c_1 = (V_{c_1}, E_{c_1})$, avec $V_{c_1} = \{1, 2, 3, 4, 5, 6\}$ et $c_2 = (V_{c_2}, E_{c_2})$ avec $V_{c_2} = \{7, 8, 9, 10, 11, 12\}$. Les Figures 1.10,1.11 et 1.12 montrent que le fait d'augmenter les arêtes entre les communautés augmente la conductance. Pour la figure 1.10, nous avons $\Phi_G = \frac{1}{2}[\frac{1}{2*15+1} + \frac{1}{2*15+1}] = 0.06$. Pour la figure 1.11, nous avons $\Phi_G = \frac{1}{2}[\frac{2}{2*15+2} + \frac{2}{2*15+2}] = 0.125$. Pour la figure 1.12, nous avons $\Phi_G = \frac{1}{2}[\frac{3}{2*15+3} + \frac{3}{2*15+3}] = 0.18$.

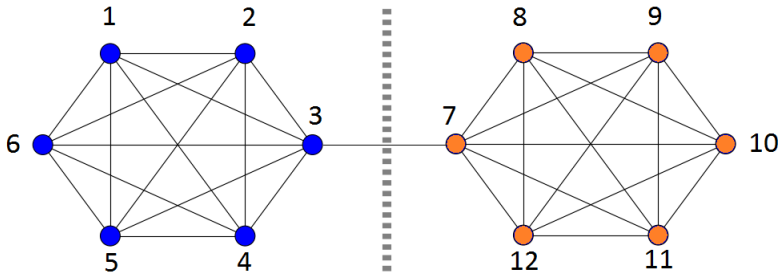


Figure 1.10 – Deux communautés reliées par une arête.

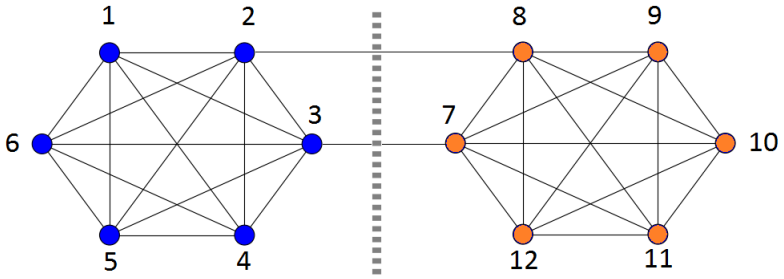


Figure 1.11 – Deux communautés reliées par deux arêtes.

La figures 1.13 montre le cas où les communautés sont connexes, en considérant un nouveau graphe $G' = (V', E')$ où $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 11, 12, 13, 14, 15, 16, 17, 18\}$, décomposé en deux communautés $c_1 = (V_{c_1}, E_{c_1})$, avec $V_{c_1} = \{1, 2, 3, 4, 5, 6, 13, 14, 15, 16\}$ et $c_2 = (V_{c_2}, E_{c_2})$ avec $V_{c_2} = \{7, 8, 9, 10, 11, 12, 17, 18\}$. La présence de nœuds isolés ne permet pas de modifier la valeur de la conductance. On ne peux donc pas utiliser la conductance dans le cas où les communautés sont non connexes.

Il existe d'autres mesures que nous avons vues auparavant, notamment la mesure de Mancoridis *et al.* (1998) ou encore la modularité que nous avons

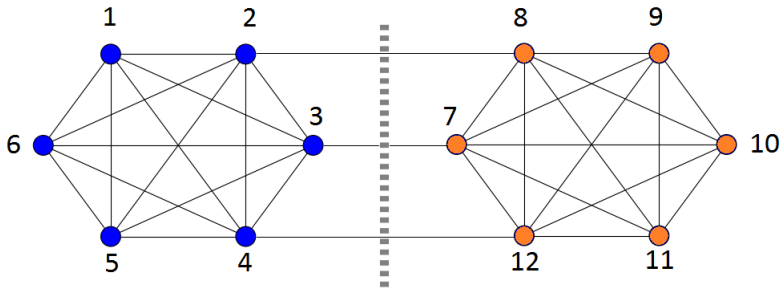


Figure 1.12 – Deux communautés reliées par trois arêtes.

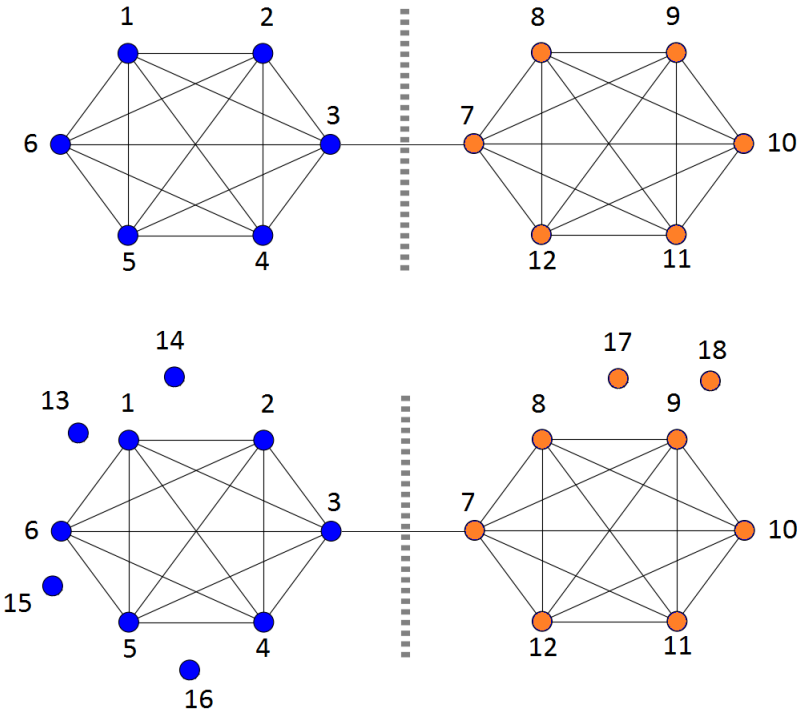


Figure 1.13 – Deux communautés reliées par trois arêtes contenant des nœuds isolés.

présentée en début de chapitre.

Récapitulatif des mesures supervisées et non supervisées

Nous avons pu observer qu'il existait de très nombreuses mesures. Nous proposons un tableau récapitulatif des méthodes avec leurs spécificités et leurs gains d'informations portant sur les communautés au sein du graphe, *Tableau 1.3*.

Mesures supervisées		
Mesures	référence	Information et caractéristiques
Rand	(Rand, 1971)	comptages sur les paires d'objets pareillement classées
Rand ajustée	(Hubert et Arabie, 1985)	amélioration de Rand
NMI	(Ana et Jain, 2003)	quantité mesurant la dépendance statistique entre deux partitions
Pureté	Manning *et al.* (2008)	taux de recouvrement maximal entre les communautés de deux partitions
F_1 score	(van Rijsbergen, 1979)	combine la précision et le rappel et leur moyenne harmonique
Mesures non supervisées		
Mesures	référence	Information et caractéristiques
Modularité	(Newman et Girvan, 2004)	maximise la densité des communautés à travers le modèle nul
Conductance	(Kannan *et al.*, 2004)	maximise la densité des communautés et minimise les liens sortants
Hamiltoniens	(Ronhovde et Nussinov, 2010)	maximise la densité des communautés à travers le modèle nul avec un facteur de résolution
MQ	Mancoridis *et al.* (1998)	exprime la somme des différences entre deux ratios de connectivités, calculés pour chaque groupe de nœud

Tableau 1.3 – Classification des mesures supervisées et non supervisées pour la détection de communautés disjointes

Dans la littérature de la détection de communautés, les mesures les plus utilisées sont les suivantes :

— la modularité
— le F_1-score
— la pureté
— la conductance
— l'indice de Rand ajusté
— le NMI

Ce sont les mesures que nous utiliserons car elles mettent en œuvre les principales caractéristiques des communautés, à savoir densité, nombre de liens sortants et graphe aléatoire. De plus, elles permettent d'établir des études comparatives dans la mesure où il s'agit des mesures les plus utilisées dans la littérature de la détection de communautés.

1.3.10 Synthèse et discussion

Ce chapitre nous a permis de formuler le problème de détection de communautés et de présenter les principales méthodes existantes dans la littérature.

Nous avons vu qu'il existait trois classes d'algorithmes :

— *les méthodes globales* qui considèrent la topologie du graphe dans son entier pour ensuite effectuer une coupe. On peut citer la méthode spectrale, les méthodes gloutonnes de déplacement de nœuds ou encore les méthodes utilisant des mesures sociales globales. En effet, la méthode spectrale demande le calcul des valeurs et des vecteurs propres dont la complexité est en $\mathcal{O}(n^3)$. Les méthodes gloutonnes de déplacement demandent d'effectuer tous les types de déplacement possibles afin d'optimiser une certaine fonction de qualité de partitionnement, comme le recuit-simulé dont la forte complexité dépend du paramétrage. Les mesures comme celles de la centralité d'intermédiarité demandent de considérer, pour toutes paires de sommets du graphe, les plus courts chemins, avec une complexité $\mathcal{O}(n^3)$. Les méthodes globales ne peuvent être appliquées à de grands graphes, tout au plus, à ceux de quelques milliers de nœuds et d'arêtes.

— *les méthodes locales* sont des méthodes dont le point de départ est le nœud et où il y a propagation d'une information ou réalisation d'une opération. Il y a par exemple propagation de labels (LPA) ou fusion d'un nœud avec un des voisins (Louvain). Ces algorithmes ont la caractéristique d'être quasi-linéaires en termes d'arête. Ce qui leur adjoint la possibilité d'être appliqués à de très grands graphes de plusieurs millions de nœuds et plusieurs centaines de millions d'arêtes. Cependant ces algorithmes sont non déterministes et hautement instables. C'est-à-dire qu'il ne retournent jamais la même réponse.

— *les méthodes hybrides* sont des méthodes locales auxquelles on associe des mesures sociales ou un gain d'information pour améliorer le résultat. Par exemple, la méthode de De Meo *et al.* (2013) utilise un graphe pondéré dont les arêtes ont été placées suivant une certaine mesure fondée sur les distances entre nœuds pour y appliquer la méthode de Louvain. Ces méthodes donnent généralement de meilleurs résultats en termes de qualité de partitionnement que les méthodes originelles. Cependant, les complexités de ces algorithmes dépendent à la fois de la complexité des méthodes locales et de la complexité des mesures sociales utilisées. Ainsi, si les mesures sociales ont une forte complexité, l'algorithme hybride ne pourra pas être appliqué sur un grand graphe.

Nous décidons d'utiliser comme base de nos propositions algorithmiques la

propagation de labels. Elle présente l'avantage d'avoir une complexité assez faible et permet de travailler sur de grands graphes. Elle a cependant certains inconvénients comme :

- une mauvaise propagation qui peut mener à de trop grandes communautés (le problème des communautés géantes).
- une instabilité
- de mauvaises propagations qui peuvent donner des communautés ayant le même label.

Ces inconvénients seront une partie des challenges que nous traîterons dans la partie algorithmique.

1.4 Détection de communautés chevauchantes

La section précédente nous a permis d'explorer les méthodes pour la détection de communautés disjointes. C'est-à-dire le cas où chaque nœud ne peut appartenir qu'à une seule communauté. Cependant, dans de nombreux modèles, des nœuds peuvent appartenir à plusieurs communautés, on parle alors de communautés chevauchantes. Par exemple, il existera des communautés chevauchantes dans un réseau de chercheurs où un auteur aura publié dans différents domaines. L'étude de ces communautés permet d'avoir une analyse plus fine du réseau. Les difficultés à surmonter sont d'une part la définition d'un nœud chevauchant, et d'autre part la manière de maîtriser la complexité algorithmique qui est plus importante que dans le cas des méthodes disjointes.

1.4.1 Formulation du problème de détection de communautés chevauchantes

Il est possible que certains nœuds appartiennent à plusieurs communautés. Par exemple, certains nœuds ont un nombre de liens identiques à plusieurs communautés. En considérant l'exemple du club de karaté présenté en introduction, où une dispute entre le manager et l'entraîneur du club a scindé le club en deux, créant deux clubs de karaté dans une même rue, on peut voir que le nœud montré par une flêche est la fois dans la communauté du manager et dans celle de l'entraineur. Il a été défini comme chevauchant dans la littérature.

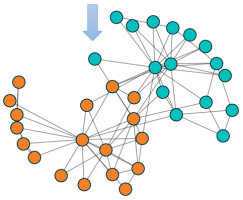


Figure 1.14 – Le graphe de Zachary avec les deux communautés et le nœud chevauchant

Des études menées par Kelley (2009), Lancichinetti *et al.* (2009), Lee *et al.* (2010), Reichardt et Bornholdt (2006), Gregory (2010), Wang *et al.* (2012) et Sales-Pardo *et al.* (2007) ont montré que le chevauchement est une caractéristique des réseaux sociaux.

En détection de communautés chevauchantes, un ensemble de communautés constituant un graphe, où plusieurs nœuds peuvent appartenir à plusieurs communautés, se nomme une *couverture* (Lancichinetti *et al.* (2009)). Ainsi, la détection de communautés chevauchantes consiste à trouver des groupes de nœuds fortement connectés entre eux et faiblement avec le reste du graphe, avec des nœuds pouvant appartenir à plusieurs communautés. Plus formellement, il s'agit de trouver une couverture $\mathcal{C} = \{C_1, ..., C_k\}$, avec $C_k \neq \emptyset$, k n'étant pas à spécifier avec $C_i \bigcap C_j = \emptyset$ ou $C_i \bigcap C_j \neq \emptyset$, et $\cup_{i=1}^{k} C_i = V$, en considérant un

graphe $G = (V, E)$.
De la définition générale que nous venons de donner résultent deux grandes grandes catégories d'algorithmes qui sont la détection de communautés chevauchantes au sens flou et la détection de communautés chevauchantes au sens de la classification multi-classe. La détection de communautés chevauchantes au sens flou porte sur l'imprécision de la classe d'appartenance d'un nœud. Dans cette catégorie de problème, on connaît déjà la classe d'appartenance de la majorité des nœuds vis-à-vis de leurs communautés. Cependant, certains nœuds situés à la frontière intérieure (cf. glossaire 6.1) de différentes communautés portent une imprécision sur leurs degrés d'appartenance quant à ces dernières. La détection de communautés chevauchantes peut être également vue comme un problème de classification multi-classe où pour tout nœud du graphe, il s'agit d'établir le degré d'appartenance à différentes communautés.

1.4.2 Détection de communautés chevauchantes à base de propagation de labels

Nous présentons les principaux algorithmes pour la détection de communautés chevauchantes à base de propagations de labels.

La première méthode connue ayant utilisé la propagation de labels fut proposée par Gregory (2010), à savoir COPRA. Durant la phase de remplacement du label d'un nœud u par un autre, les auteurs proposent d'utiliser un vecteur pour maintenir les labels les plus communs avec l'intervention d'un seuil de probabilité. Le résultat, grâce à cette liste, est qu'un nœud peut appartenir à une ou plusieurs communautés. Bien que cette méthode ait une complexité linéaire en termes d'arêtes, elle nécessite de spécifier le nombre de communautés (ν) à laquelle pourrait appartenir un nœud. Cette méthode est également instable, ne donnant que très rarement la même partition d'un lancement à l'autre. L'instabilité nécessite de lancer plusieurs fois l'algorithme avant d'obtenir une solution correcte en termes de qualité. Cette méthode est sujette aux mauvaises propagations de labels, donnant des communautés géantes. La Figure 1.15 donne un exemple de résultatde COPRA avec $\nu = 2$.

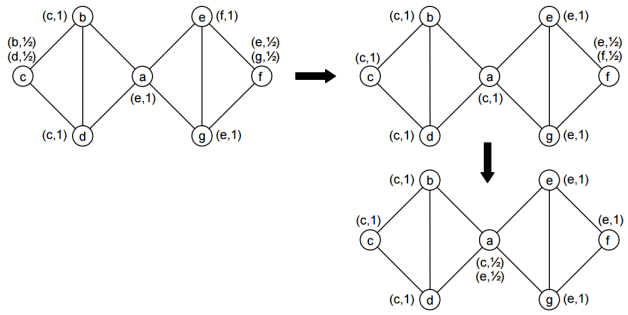


Figure 1.15 – Exemple d'application de COPRA (extrait de Gregory (2010))

SLPA pour "*Speakerlistener Label Propagation Algorithm*" (Xie *et al.* (2011)) est un algorithme fondé sur une mémoire de labels et sur la popularité des labels les plus influents au sein du graphe, c'est-à-dire, des labels que l'on retrouve le plus souvent pour l'identification des communautés résultantes du graphe. Le changement de label d'un nœud n'est plus fonction du voisinage actuel, mais dépend de la mémoire du nœud dont les labels voisins ont changé au cours du temps. Initialement, on attribue à chaque nœud une mémoire de labels. Il s'agit d'un vecteur de taille k (si le nœud courant a k voisins) qui est originellement constitué des labels de ses nœuds voisins. Deux types de nœuds sont définis dans l'algorithme SLPA, les *speakers* et *listeners*. Un nœud est alors sélectionné aléatoirement comme speaker et propage son label pendant que les autres observent la propagation, jusqu'à stabilisation. C'est alors qu'un autre nœud du graphe est sélectionné pour jouer le rôle du speaker. Une fois le processus de propagation terminé, on peut observer la force d'appartenance qu'a un nœud vis-à-vis des diverses communautés via l'influence des nœuds dans les différentes mémoires. A la fin du processus, on regarde toutes les mémoires de tous les nœuds pour y voir les labels des communautés les plus fréquentes. Pour sélectionner les labels les plus fréquents, on utilise un seuil $r \in [0, 1]$ qui permet de considérer les labels les plus influents (ceux les plus fréquents dans la mémoire). Il peut y a voir des labels ayant le même score d'influence, permettant ainsi le chevauchement. Les résultats en termes de qualité de partitionnement sont encourageants, la majorité des nœuds chevauchants dans les graphes utilisés dans leurs expérimentations sont détectés comme pour le club de karaté, le réseau footballistique , le réseau de livres scientifiques, le réseau de collaboration scientifique et d'autres graphes sociologiques. Cependant, l'instabilité persiste avec la possibilité qu'il y ait de mauvaises propagations, surtout si elles ont lieu au début de l'algorithme et influent sur la suite du processus. On peut également se questionner sur la méthode employée, dans la mesure où les communautés détectées lorsque qu'un speaker envoie son label sont disjointes. Ce sont les structures communautaires qui empêchent la propagation du label du speaker de continuer, mais pas forcément les nœuds qui peuvent appartenir à plusieurs communautés. Le paramètre r a également une influence forte sur le pourcentage de nœuds chevauchants. L'algorithme est également assez long car il nécessite d'obtenir la stabilisation, c'est-à-dire que les communautés détectées ne puissent plus évoluer.

BMLPA pour "*balanced multi-label propagation algorithm*" (Wu *et al.* (2012)) est une amélioration de COPRA. Les auteurs proposent une stratégie de mise à jour, qui demande que les nœuds ayant le même label aient un cœfficient d'appartenance vis-à-vis d'autres communautés. Cela a pour conséquence que certains nœuds puissent appartenir à plusieurs communautés. Les auteurs proposent également de générer des "cœurs bruts", qui sont utilisés pour l'initialisation des vecteurs pour la propagation de labels multiples. Cela permet d'aller plus vite et de stabiliser une partie du graphe lors de la propagation. Les résultats en termes de qualité de partionnement sont encourageants et permettent de limi-

ter le nombre de mauvaises propagations (phénomène de communautés géantes) sans toutefois les faire disparaître.

MLPA pour *multi-label propagation algorithm* (Dai *et al.* (2013)) utilise une propagation de labels avec intensité qui permet d'attribuer une force aux labels des nœuds émettants. L'idée est que lorsqu'un nœud mettra à jour son label, il aura connaissance des labels des nœuds voisins, et également des zones sur lesquelles ils se sont déjà propagés , qui sont en fait, les communautés en cours de création. Dès lors, certaines propagations peuvent s'arrêter si l'intensité du label en question est trop forte ou au contraire être mises en exergue si son intensité est trop faible. L'utilisation de l'intensité des nœuds permet de diriger la propagation de labels pour éviter de mauvaises propagations (impossibilité de détecter de trop petites commununautés). Cette méthode a comme force de donner de l'information sur les structures topologiques qui se créent au fur et à mesure du développement de l'algorithme. Cependant, c'est une faiblesse, notamment pour des graphes dont la densité est élevée, où l'intensité devra être ajustée pour continuer des propagations de labels non finies.

Les méthodes présentées ci-dessus se fondent soit sur une modification de la propagation de labels en incluant un facteur d'appartenance, soit sur une multi-liste de propagations de labels. Cependant, cela ne réduit pas l'instabilité et des paramètres doivent être évalués pour connaître le pourcentage de nœuds chevauchants et le nombre de communautés auxquelles appartient un nœud.

1.4.3 Autres méthodes pour la détection de communautés chevauchantes

Il existe d'autres méthodes pour la détection de communautés chevauchantes, fondées sur la topologie du graphe et le nombre de connexions qu'un nœud a vis-à-vis des communautés avoisinantes.

Palla *et al.* (2005) ont créé *Cfinder*, un algorithme pour la détection de communautés chevauchantes fondé sur la recherche de motifs locaux, par percolation de cliques (CPM : Clique Percolation Method). Les auteurs observent qu'une communauté peut-être définie comme une chaîne de k-cliques adjacentes (cf. glossaire 6.1). Cette méthode permet la détection de communautés couvrantes où un sommet peut appartenir à plusieurs k-cliques. Les résultats en termes de qualité de partitionnement sont encourageants mais présentent trois inconvénients majeurs. Le premier est la nécessité de paramétrer k (c'est-à-dire la taille des cliques). Le second concerne la complexité algorithmique. Il a été démontré que la complexité pour détecter les 3-cliques (Berge (1958)) d'un graphe était en $\mathcal{O}(n^{1.41})$ (n étant le nombre de sommets), mais ce nombre augmente de manière non linéaire en fonction du nombre de sommets. Le troisième inconvénient est la recherche de motifs statiques qui n'est pas un trait absolu

des réseaux complexes (avec des nœuds chevauchants des communautés de taille et de topologie différentes). Les expérimentations menées pas Palla *et al.* (2005) ont montré que la valeur $k = 4$ donnait les résultats en termes de qualité de partitionnement les plus probants. Cette méthode peut selon la paramétrisation de k être appliquée à des graphes de plusieurs centaines de milliers d'arêtes. Nous donnons un exemple tiré du site de Gergely Palla à la Figure 1.16 où les nœuds de couleur rouge appartiennent à des communautés chevauchantes.

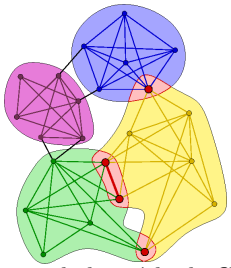


Figure 1.16 – Exemple d'application de la méthode CFinder (Extrait du site de Palla *et al.* (2005))

Zhang *et al.* (2007) proposent un algorithme spectral pour la détection de communautés chevauchantes. Le principe est de calculer un certain nombre de vecteurs propres liés à la matrice Laplacienne représentant le graphe, puis d'appliquer sur cet espace propre un algorithme flou de clustering, le Fuzzy C means (FCM) et de retranscrire les résultats sur le graphe pour obtenir les recouvrements. La méthode nécessite de spécifier le nombre de recouvrements et nécessite le calcul des valeurs et des vecteurs propres. Les résultats en termes de qualité sont encourageants. L'algorithme a une complexité en $\mathcal{O}(n^3 + ndc^2i)$, avec d, le nombre de dimensions, c le nombre de clusters et i le nombre d'itérations.

Psorakis *et al.* (2011) proposent un modèle fondé sur la factorisation par matrices non négatives bayésiennes (NMF). Cet algorithme nécessite de fournir le nombre K de recouvrements et demande de très nombreux calculs matriciels. La complexité de cet algorithme est en $\mathcal{O}(Kn^2)$.

Shen *et al.* (2009) proposent EAGLE (agglomerativE hierarchicAl clusterinG based on maximaL cliquE). EAGLE est une méthode agglomérative fondée sur la détection de cliques maximales (cf. Glossaire 6.1) et l'optimisation locale d'une fonction de qualité permettant l'élaboration d'un dendrogramme. L'algorithme opère en deux étapes sur le graphe. Premièrement, les cliques maximales sont trouvées pour former les premières communautés. La seconde étape consiste à fusionner les communautés ayant la plus grande similarité. La fonction de similarité n'est autre que la modularité de Newman. C'est sur le nœud où portera la fusion qu'il y aura chevauchement. La coupe optimale est donnée en utilisant une version chevauchante de la modularité de Newman, la modularité de Shen que nous détaillerons dans la partie concernant les mesures non supervisées pour

la détection de communautés chevauchantes. EAGLE donne des résultats satisfaisants en termes de qualité de recouvrement. Cependant, dans certains cas, les nœuds qui peuvent être classés comme chevauchants et qui sont liés à des communautés faiblement denses ne sont pas détectés. Cette méthode peut soulever certaines questions quant aux strutures communautaires chevauchantes. On peut reprocher à cet algorithme d'être à la recherche de motifs plutôt que d'être à la recherche de nœuds entre communautés. Des structures communautaires peuvent exister sans être des cliques. L'algorithme présente également une complexité assez élevée en $\mathcal{O}(n^2 + s(h+n))$ avec s le nombre de cliques maximales et h le nombre de paires de cliques maximales qui sont voisines (partageant un nœud en commun).

Lee *et al.* (2010) ont proposé GCE (Greedy Clique Expansion). Cet algorithme est très similaire à EAGLE. Il identifie dans un premier temps les cliques comme des graines (leaders) au sein du graphe. Ces graines, qui sont des cœurs de communautés s'agrandissent par optimisation d'une fonction de qualité locale. Les auteurs choisissent la fonction de fitness de Lancichinetti *et al.* (2009), à savoir pour une communauté S, $F_S = \frac{k_{in}^S}{(k_{in}^S + k_{out}^S)^\alpha}$ où α est un paramètre ajustable, k_{in}^S (nombre d'arêtes internes dans S) et k_{out}^S (nombre d'arêtes ayant une extrémité hors de S). Cette mesure privilégie une expansion vers des nœuds maximisant les liens internes et minimisant les liens externes. Lorsque deux structures communautaires veulent effectuer une agglomération sur un nœud spécifique, ce nœud est classé comme chevauchant. La complexité de cet algorithme est en $\mathcal{O}(mh)$ avec m, le nombre d'arêtes et h le nombre de cliques maximales détectées initialement et considérées comme graines. Les résultats expérimentaux montrent que la qualité de recouvrement est légèrement meilleure que celle d'EAGLE, notamment sur les graphes artificiels LFR (cf. glossaire 6.1).

Lancichinetti *et al.* (2011) ont proposé OSLOM (Order Statistics Local Optimization Method). Il s'agit d'une méthode de raffinage agissant sur une partition déjà fournie. OSLOM utilise les algorithmes de Louvain ou d'Infomap pour créer une première partition. C'est alors que la méthode ajoute ou retire des nœuds pour arriver à un état stable où il n'est plus intéressant de modifier la structure topologique des communautés. Le principe consiste à comparer les structures communautaires entre le modèle nul (cf. glossaire 6.1) et la partition réelle en se fondant sur le nombre de liens dans les communautés et sur le degré du nœud étudié. Si le nœud étudié a plus de connexions vis-à-vis d'une communauté que la valeur moyenne issue du modèle nul, ce nœud sera ajouté à cette communauté. Il sera retiré dans le cas contraire. Le processus se répète sur tous les nœuds du graphe en effectuant des modifications sur la partition originelle si certains groupes de nœuds n'ont pas la caractéristique de structures communautaires. L'algorithme est non déterministe car il dépend de l'ordre de visite selon lequel est appliquée cette méthode.

L'algorithme comporte quatre paramètres dont un seuil de probabilité à

donner. L'algorithme est assez lent et est fonction du nombre de nœuds. Il peut nécessiter plusieurs journées notamment sur de grands graphes comme Live-Journal. La paramétrisation standard donne cependant des résultats corrects en termes de qualité pour les communautés résultantes.

1.4.4 Tableau récapitulatif des méthodes chevauchantes

L'état de l'art concernant la détection de communautés chevauchantes a montré l'existence de très nombreuses méthodes. Nous récapitulons les caractéristiques concernant leur complexité et la taille des réseaux sur lesquels ces solutions peuvent être appliquées dans la mesure où nous souhaitons traîter de grands graphes de plusieurs millions d'arêtes.

Algorithmes LPA	articles	dendrogramme	complexité (ordre)	parallélisé
COPRA	Gregory (2010)	non	$\mathcal{O}(kmn)$	non
SLPA	Xie *et al.* (2011)	non	$\mathcal{O}(2n + n^2m)$	non
BLMPA	Wu *et al.* (2012)	non	$\mathcal{O}(kmn)$	non
MLPA	Dai *et al.* (2013)	non	$\mathcal{O}(kmn)$	non
EAGLE	Shen *et al.* (2009)	non	$\mathcal{O}(n^2 + (h+n)s)$	non
NMF	Psorakis *et al.* (2011)	non	$\mathcal{O}(Kn^2)$	non
FCM	Zhang *et al.* (2007)	non	$\mathcal{O}(n^3 + ndc^2i)$	non
OSLOM	Lancichinetti *et al.* (2011)	non	"*"	non
CFINDER	Adamcsek *et al.* (2006)	non	$\mathcal{O}(e^n)$	non

Tableau 1.4 – Ordre de complexité des algorithmes chevauchants. Le signe "*" signifie que la complexité est difficile à établir. n : nombre de sommets, m : nombre d'arêtes, k : paramêtre (entier), i : nombre d'itérations, d : le nombre de dimensions de l'espace propre et c : le nombre de clusters

La comparaison des figures 1.17 et 1.9 permet d'affirmer que les méthodes chevauchantes ont une complexité plus élevée que les méthodes disjointes.

D'après nos expérimentations et la taille des graphes sur lesquels ont été appliqués les algorithmes de la figure 1.17, nous pouvons constater que certains algorithmes peuvent être appliqués sur des graphes d'assez grandes tailles alors que ce n'est pas le cas pour d'autres. Par exemple, OSLOM peut-être appliqué sur une machine à des graphes de plus 18 millions de sommets et 300 millions d'arêtes (il s'agit du réseau web du Royaume-Uni (SNAP)) alors les autres ne peuvent pas le traîter. D'après nos expérimentations et en considérant comme critère l'erreur mémoire, l'algorithme permettant de traîter les plus grands graphes est OSLOM. Les méthodes ne permettent pas de traiter les graphes de plus de 10 millions d'arêtes.

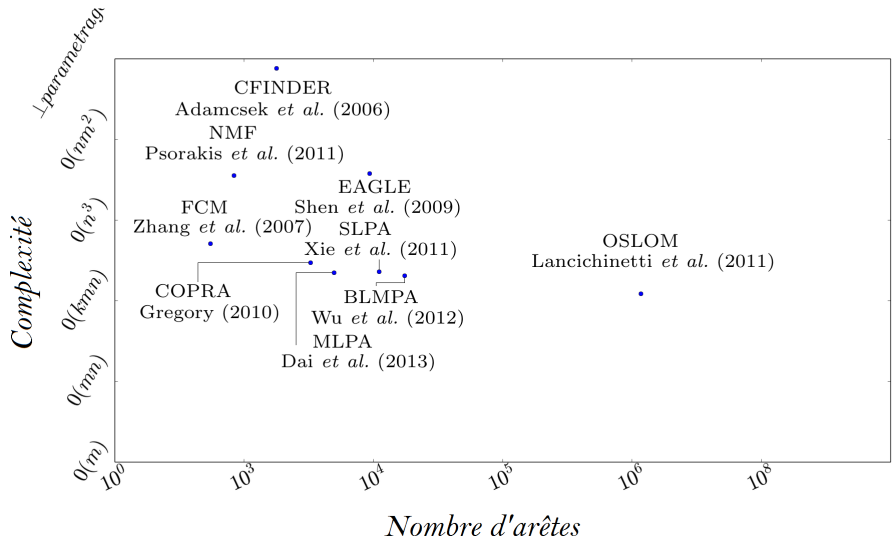


Figure 1.17 – Méthodes pour la détection de communautés chevauchantes. L'axe des abscisses représente la taille des graphes, l'axe des ordonnées, la complexité.

1.4.5 Mesures supervisées et non supervisées pour la détection de communautés chevauchantes

Certaines mesures supervisées et non supervisées dans le cas de la détection de communautés disjointes ont été retranscrites pour le cas du chevauchement. Nous nous proposons d'exposer les mesures les plus utilisées dans la littérature ainsi que leurs spécificités. A la fin de cette section, nous présenterons un tableau récapitulatif des mesures.

Mesures supervisées pour la détection de communautés chevauchantes

L'information mutuelle normalisée pour le chevauchement fut proposée par Lancichinetti *et al.* (2009) . Considérons deux couvertures $\mathcal{A}$ et $\mathcal{B}$ de V. La probabilité de tirer un élément et qu'il appartienne à une partie de la partition ($A \in \mathcal{A}$) est $\frac{n_A}{n}$, où n_A est le nombre d'éléments dans A. L'entropie de Shanon de la partition $\mathcal{A}$ est ainsi définie par :

$$H(\mathcal{A}) = -\sum_{A \in \mathcal{A}} \frac{n_A}{n} log_2 \frac{n_A}{n} \tag{1.23}$$

Lancichinetti *et al.* (2009) ont proposé une version du NMI pour la comparaison de couvertures $\mathcal{A}$ et $\mathcal{B}$ comme étant :

$$NMI_{LFK} = 1 - \frac{1}{2}\left(\frac{H(\mathcal{A}|\mathcal{B})}{\mathcal{A}} + \frac{H(\mathcal{A}|\mathcal{B})}{\mathcal{B}}\right) \tag{1.24}$$

avec $H(\mathcal{A}|\mathcal{B})$ étant l'entropie conditionnelle. Cette quantité mesure l'entropie restante provenant de la couverture B, si l'on connait parfaitement la seconde couverture A. $H(\mathcal{A}|\mathcal{B}) = 0$ si et seulement si la couverture B est complètement déterminée par la couverture A. La valeur de NMI_{LFK} est comprisentre 0 et 1. Plus cette valeur est proche de 1, plus les deux convertures sont identiques.

McDaid *et al.* (2011) ont proposé une extension au NMI de Lancichinetti et al. avec l'intervention d'une pénalité si les deux couvertures sont trop différentes. Les résultats des deux mesures sont équivalents.

L'indice d'Omega (Collins et Dent (1988)) est fondée sur des paires de nœuds qui sont dans les mêmes communautés selon les deux couvertures.

Considérons deux couvertures P_1 et P_2, l'indice d'omega est défini de la manière suivante :

$$\Omega(P_1, P_2) = \frac{o_u(P_1, P_2) - o_e(P_1, P_2)}{1 - o_e(P_1, P_2)} \tag{1.25}$$

où $o_u(P_1, P_2)$ représente la fraction de paires de nœuds qui apparaissent ensemble dans les mêmes communautés à la fois dans P_1 et P_2.
$o_u(P_1, P_2) = \frac{2}{n(n-1)} \sum_j |t_j(P_1) \cap t_j(P_2)|$ où $t_j(P)$ est l'ensemble des paires de nœuds qui apparaissent dans exactement j communautés dans la couverture P. Le nombre de paires de nœuds est égale à $\frac{n(n-1)}{2}$. Le terme $o_e(P_1, P_2)$ est la valeur espérée de cette fraction dans le modèle nul.

$$o_e(P_1, P_2) = \frac{4}{n^2(n-1)^2} \sum_j |t_j(P_1)||t_j(P_2)| \tag{1.26}$$

L'indice d'Omega prend sa valeur entre 0 et 1. Une valeur proche de 1 signifie que les deux couvertures sont identiques. Dans la partie expérimentale, nous utiliserons la notation Ω pour signaler l'indice d'Omega.

Le F_1-score pour le chevauchement se définit de la même manière que pour le cas disjoint, à savoir :

$$F_1 = \frac{2 \times précision \times rappel}{précision + rappel} \tag{1.27}$$

où le rappel est le nombre de bons nœuds chevauchants détectés divisé par le véritable nombre de nœuds chevauchants. La précision est le nombre de bons nœuds chevauchants détectés divisé par le nombre total de nœuds chevauchants détectés. Le F_1-score est un bon compromis entre la qualité de nœuds chevauchants bien détectés et la qualité de ces détections. Elle atteint sa valeur maximale à 1 et sa plus basse à 0.

Mesures non supervisées pour la détection de communautés chevauchantes

Plusieurs extensions de la modularité ont été proposées pour le cas du chevauchement. Nous proposons de les exposer en les détaillant. Elles se basent toutes sur un cœfficient d'appartenance entre un nœud i et une communauté c, que l'on note $a_{i,c}$. On considère par la suite une couverture de communautés chevauchantes $C = \{c_1, ..., c_{|C|}\}$ et un vecteur d'appartenance pour un nœud i à chacune des communautés $(a_{i,c_1}, ..., a_{i,c_{|C|}})$. Le cœfficient d'appartenance d'un nœud i est régi par deux contraintes qui sont $0 \leq a_{i,c} \leq 1, \forall i \in V, \forall c \in C$ et par $\sum_{c \in C} a_{i,c} = 1$.

Zhang *et al.* (2007) ont proposé une extension de la modularité qui utilise la moyenne du cœfficient d'appartenance entre deux nœuds pour mesurer la qualité des structures communautaires chevauchantes :

$$Q_{Ov}^{Z} = \sum_{c \in C} \frac{|E_c^{in}|}{|E|} - (\frac{2|E_c^{in}| + |E_c^{out}|}{2|E|})^2 \tag{1.28}$$

avec $|E_c^{in}| = \frac{1}{2} \sum_{i,j \in c} \frac{a_{i,c}+a_{j,c}}{2} A_{ij}$, $|E_c^{out}| = \sum_{i \in c, j \notin c} \frac{a_{i,c}+(1-a_{j,c})}{2} A_{ij}$ et $|E| = \frac{1}{2} \sum_{i,j} A_{ij}$. La méthode proposée par Zhang pénalise les structures détectées ayant un nombre de liens trop fort en dehors de ces dernières. Les résultats en termes de qualité pour des graphes dont les communautés présentent des densités équivalentes sont bons, mais la qualité se détériore si les structures communautaires sont trop différentes.

Nepusz *et al.* (2008) ont considéré le cœfficient d'appartenance $a_{i,c}$ comme la probabilité qu'un nœud i soit dans la communauté c. Ainsi, la probabilité que le nœud i appartienne à la même communauté que j est le produit scalaire de leur vecteur d'appartenance noté $s_{ij} = \sum_{c \in C} a_{i,c} a_{j,c}$. Les auteurs ont également considéré s_{ij} comme une mesure de similarité entre les nœuds i et j. Ainsi, en remplaçant le symbole de Kronecker dans la formule générale de la modularité, les auteurs définissent leur modularité comme étant :

$$Q_{Ov}^{Nep} = \frac{1}{2|E|} \sum_{i,j} [A_{ij} - \frac{k_i k_j}{2|E|}] s_{ij} \tag{1.29}$$

La formule est sujette au degré des nœuds chevauchants. C'est-à-dire que si la distribution des degrés des nœuds ne suit pas une loi uniforme, la mesure aura plus de difficulté a détecter les nœuds chevauchants. Les résultats en termes de qualité sur les graphes réels sont cependant encourageants.

Nicosia *et al.* (2009) ont fait une extension à la modularité pour le cas du chevauchement entre communautés. Les auteurs la définissent de la manière

suivante :

$$Q_{Ov}^{Ni} = \frac{1}{m} \sum_{c} \sum_{i,j \in V} [\beta_{ij,c} A_{ij} - \beta_{ij,c}^{out} \beta_{ij,c}^{in} \frac{k_i^{out} k_j^{in}}{m}] \tag{1.30}$$

où $\beta_{ij,c}$ est le cœfficient d'appartenance du lien ij à la communauté c, $\beta_{ij,c}^{out}$ est le cœfficient d'appartenance espéré de tous les liens possibles ij du nœud i au nœud j à l'intérieur de la communauté c et $\beta_{ij,c}^{in}$ est le cœfficient d'appartenance de n'importe quel lien ij pointant vers un nœud j dans la communauté c. Les termes k_i^{out} et k_i^{in} sont respectivement le degré sortant et entrant du nœud i. Il est à noter que $\beta_{ij,c}$ est une fonction de la forme $\beta_{ij,c} = F(a_{i,c}, a_{j,c})$ avec $a_{i,c}$ cœfficient d'appartenance du nœud i par rapport à la communauté c. $F(a_{i,c}, a_{j,c}) = \frac{1}{(1+e^{-f(a_{i,c})})(1+e^{-f(a_{j,c})})}$ où $f(a_{i,c})$ est une fonction linéaire échelonnable du type $f(x) = 2px - p$, $p \in \mathbb{R}$. Les auteurs ont proposé d'utiliser la fonction $f(x) = 60x - 30$ pour le paramétrage de la modularité chevauchante car elle donnait des résultats très encourageants. Il s'agit également de la paramétrisation standard dans la littérature et que nous utiliserons pour nos expérimentations. La valeur de la modularité de Nicosia varie entre 0 et 1. Une valeur proche de 0 implique que l'algorithme n'a détecté qu'une seule communauté (le graphe en lui même) alors qu'une valeur proche de 1 signifie la présence de structures communautaires possiblement chevauchantes.

Shen *et al.* (2009) ont proposé une modularité avec un cœfficient d'appartenance comme étant l'inverse du nombre de communautés auxquelles pourrait appartenir un nœud, $a_{i,c} = \frac{1}{O_i}$ où O_i est le nombre de communautés contenant le nœud i. La version qu'ont proposée les auteurs donne la formule :

$$Q_{ov}^{Shen_1} = \frac{1}{2|E|} \sum_{c \in C} [\sum_{i,j \in c} (\frac{k_i k_j}{|E|})] \frac{1}{O_i O_j} \tag{1.31}$$

Les résultats sont encourageants mais peuvent être mauvais si le nombre de communautés auxquelles appartient un nœud par rapport à un autre est trop important. Ainsi, certains bons chevauchements risquent de ne pas être détectés. Par la suite, les mêmes auteurs ont proposé une nouvelle modularité pour le chevauchement en se fondant sur la présence de sous-graphes complets, un nombre élevé de cliques et par une modification du cœfficient d'appartenance d'un nœud aux communautés auxquelles il pourrait appartenir. Le cœfficient devient $a_{i,c} = \frac{1}{a_i} \sum_{k \in c} \frac{M_{ik}^c}{M_{ik}} A_{ik}$ où M_{ik} représente le nombre de cliques maximum dans un réseau contenant l'arête (i,k) et $a_i = \sum_{c \in C} \sum_{k \in c} \frac{M_{ik}^c}{M_{ik}} A_{ik}$, avec M_{ik}^c représentant le nombre de cliques maximum dans la communauté c contenant l'arête (i,k). Cette dernière formule donne en termes de qualité de partitionnement pour le chevauchement de meilleurs résultats que celle présentée auparavant. Sa valeur varie entre 0 et 1, et est sensée être la meilleure réponse pour une valeur proche de 1. Elle présente comme avantage qu'un nœud peut appartenir à des communautés de tailles différentes.

Chen *et al.* (2013) ont proposé la densité de modularité dont l'objectif est de favoriser les petites structures communautaires pour contourner le problème de résolution de limite, c'est-à-dire que si il existe des communautés de tailles différentes à l'intérieur d'un même graphe, certaines communautés, même bien définies, pourront ne pas être distinguées dans la partition de modularité optimale. Pour ce faire, les auteurs font intervenir une *pénalité de coupe* qui est simplement la fraction d'arêtes qui connecte les nœuds à d'autres communautés. La densité de modularité est définie par :

$$Q_{ds}^{Chen1} = \sum_{c\in C}[\frac{|E_c^{in}|}{|E|}d_c - (\frac{2|E_c^{in}| + |E_c^{in}|}{2|E|}d_c)^2 - \sum_{c'\in C, c'\neq c}\frac{|E_{c,c'}|}{2|E|}d_{c,c'}] \quad (1.32)$$

avec $d_c = \frac{2|E_c^{in}|}{|c|(|c|-1)}$ et $d_{c,c'} = \frac{|E_{c,c'}^{in}|}{|c|(|c'|)}$. Les résultats en termes de qualité sont bons sur des réseaux réels mais présentent un inconvénient notamment pour les graphes faiblement denses. Cela est d'autant plus vrai que le nombre de communautés avec une forte densité sera important. En 2015, les mêmes auteurs (Chen *et al.* (2015)) ont amélioré leur méthode en y incluant le nombre de liens sortant des structures communautaires détectées.

$$Q_{ds}^{Chen2} = \sum_{c\in C}[\frac{|E_c^{in}|}{|E|}d_c - (\frac{2|E_c^{in}| + |E_c^{in}|}{2|E|}d_c)^2 - \sum_{c'\in C, c'\neq c}\frac{|E_{c,c'}|}{2|E|}d_{c,c'}] \quad (1.33)$$

avec dorénavant $d_c = \frac{2|E_c^{in}|}{\sum_{i,j\in c, j\in c'} f(a_{i,c},a_{j,c})}$ et $d_{c,c'} = \frac{|E_{c,c'}^{in}|}{\sum_{i\in c, j\in c'} f(a_{i,c},a_{j,c'})}$. $|E_c^{in}| = \frac{1}{2}\sum_{i,j\in c} f(a_{i,c},a_{j,c})A_{ij}$,$|E_c^{out}| = \sum_{i\in c}\sum_{c'\in c, c'\neq c, j\in c'} f(a_{i,c},a_{j,c'})A_{ij}$, $|E_{c,c'}| = \sum_{i\in c, j\in c'} A_{ij}$ et$|E|$ le nombre d'arêtes du graphe.
La fonction $f(a_{i,c},a_{j,c'})$ peut être le produit ou la moyenne des vecteurs $a_{i,c}$ et $a_{j,c}$. Sa valeur varie entre 0 et 1, et donne la meilleure réponse pour une valeur proche de 1. Les résultats sont meilleurs que pour leur première version mais la formule proposée reste fondée sur une moyenne d'appartenance. C'est-à-dire que si un nœud est susceptible d'appartenir à plusieurs communautés, il aura plus de chance de considérer les communautés avec lesquelles il aura beaucoup de liens, ce qui peut être problématique pour les réseaux complexes, avec des tailles de communautés différentes.

Récapitulatif des mesures supervisées et non supervisées

Nous proposons un tableau récapitulatif de toutes les mesures citées précédemment avec leurs informations relatives en prenant comme source d'information les articles en question. Le tableau 1.5 nous montre l'existence de très nombreuses mesures. Il y en a d'ailleurs d'autres.

Cependant, dans la littérature de la détection de communautés, les mesures les plus utilisées sont :

Mesures supervisées (cas du chevauchement)		
Mesures	référence	Information et caractéristique
Omega	(Collins et Dent, 1988)	comptages sur les paires disjointes et chevauchantes d'objets pareillement classés
NMI	(Lancichinetti *et al.*, 2009)	quantité mesurant la dépendance statistique entre deux couvertures.
F_1-score	(Yang et Leskovec, 2013)	combine la précision et le rappel avec leur moyenne harmonique (cf. glossaire 6.1) en tenant compte des chevauchements.
Mesures non supervisées (cas du chevauchement)		
Mesures	référence	Information et caractéristique
$Q_{Nicosia}$	(Nicosia *et al.*, 2009)	maximise la densité des communautés à travers le modèle nul
Q_{Shen}	Shen *et al.* (2009)	
Q_{Chen}	(Chen *et al.*, 2013)	
Q_{Nepusz}	(Nepusz *et al.*, 2008)	

Tableau 1.5 – Classification des mesures supervisées et non supervisées pour la détection de communautés

— la modularité pour le chevauchement ((Nicosia *et al.*, 2009))
— le F_1-score pour le chevauchement
— le NMI pour le chevauchement
— l'indice d'omega

Ce sont les mesures que nous utiliserons car elles mettent en œuvre les principales caractéristiques des communautés, à savoir la densité, le nombre de liens sortants et la comparaison entre structures détectées avec un graphe aléatoire.

1.4.6 Synthèse et discussion

Cette section a permis de donner notre formulation du problème de détection de communautés chevauchantes, d'analyser les méthodes existantes et de voir les mesures de qualité qui y sont associées. Dans la mesure où nous souhaitons créer une méthode de chevauchement stable (c'est-à-dire donnant quasi-systématiquement les mêmes résultats) et destinée à des graphes de grandes tailles, nous portons notre synthèse sur la complexité, le degré de paramétrage de ces méthodes et la stabilité.

Les méthodes chevauchantes présentent des complexités plus élevées que les méthodes disjointes. Cela s'explique du fait qu'une partie des algorithmes chevauchants repose sur le fait de lancer plusieurs fois un algorithme disjoint et d'observer pour un nœud les communautés auxquelles il appartient le plus (comme COPRA). Malheureusement, ces algorithmes demandent un espace de stockage assez fort, dépassant les capacités d'une seule machine.

Les méthodes chevauchantes présentent également un degré de paramétrage plus élevé. Pour les algorithmes robustes à base de méthodes disjointes, cela exige de connaître le nombre d'algorithmes à lancer en parallèle. De nombreux algorithmes nécessitent également de donner un nombre de communautés auxquelles un nœud pourrait appartenir, ce qui nécessite des études préalables.

La plupart des algorithmes de détection de communautés chevauchantes sont instables. Cela est dû en partie à ce que beaucoup d'algorithmes reposent sur des algorithmes non déterministes disjoints.

Cependant, les méthodes par propagation de labels ont été l'objet de nombreuses études et constituent, pour l'étude des grands graphes, l'option à privilégier.

Chapitre 2

Parallélisme et distribution

Ce chapitre décrit dans un premier temps les principaux outils pour la programmation et la gestion de graphes dans un domaine parallèle et distribué. Dans un second temps, les algorithmes pour la détection de communautés dans un environnement parallèle et distribué sont présentés. Nous donnerons à la fin de ce chapitre un tableau récapitulatif des méthodes de détection de communautés dans le domaine parallèle et distribué et nous discuterons l'axe à choisir pour nos objectifs de recherche.

2.1 Problématique de la détection de communautés dans de grands graphes

Notre société moderne produit un nombre de données de plus en plus important. Des années 1970 avec le "minitel" dont les mémoires moyennes étaient de l'ordre de 8,25 Ko (exemple du minitel 1 bistandard Alcatel/Telic.) au datacenter (417 600 serveurs à Douglas County, 204 160 serveurs à Dallas ou 241 280 serveurs à Council Bluffs d'après le site "http ://www.artificialbrains.com/google/datacenters") de nos jours qui peut stocker plusieurs centaines de milliers de teraoctets et consomme en moyenne 103 MW (selon le journal "The Guardian"), soit autant que la ville anglaise de Newcastle, le monde a connu une évolution en informatique sans précédent. Les ensembles de données devenant de plus en plus volumineux, il est difficile de travailler avec des outils classiques de gestion de bases de données ou de gestion de l'information. Cela nous amène à la notion de données massives.

Aucune définition précise ou universelle ne peut être donnée du Big Data. Il s'agit d'un concept de stockage d'un nombre très important d'informations sur une base numérique, utilisant plusieurs machines qui forment un *cluster*, et les algorithmes peuvent être utilisés de manière parallèle et distribuée. La gestion des données massives marque une rupture dans l'évolution des systèmes

d'information et répond à une quintuple exigence, on parle de la règle des "5V" :

- grand *V*olume de données (qui ne peut généralement être stocké sur une seule machine)
- importante *V*ariété de ces mêmes données.
- *V*itesse de traitement avoisinant le temps réel.
- *V*éracité : La véracité ou fiabilité des données (s'assurer que les données soient réelles et non faussées).
- *V*aleur : Dans un contexte de surcharge informationnelle, il s'agit de sélectionner l'information pertinente pour l'obtention d'un meilleur résultat.

L'analyse de grands graphes est un exemple de problématique. Par exemple, la librairie igraph en python, permettant l'analyse de graphes, ne peut charger un graphe dont la capacité de fichier représentant les arêtes excède 60 Mo. Cela pose un problème majeur dans la mesure où de plus en plus de graphes ont des tailles de fichier très importantes. Ainsi, les graphes représentant Twitter, Facebook, You Tube ou encore Amazon ne peuvent être étudiés. Une autre limite concerne le côté algorithmique de l'analyse des grands graphes. Certains algorithmes ont une complexité bien trop grande pour pouvoir donner une réponse en un temps acceptable. C'est ainsi que naît le besoin de distribuer et de répartir les données sur plusieurs machines pour y appliquer un algorithme parallèle, donnant une réponse en un temps acceptable. C'est pourquoi depuis ces trente dernières années des plateformes sont apparues permettant le développement de manière parallèle et distribuée.

2.2 Plateformes parallèles et distribuées

Pour traiter de grandes quantités de données, des outils permettant la parallélisation d'algorithmes et la distribution des données ont été proposés. Nous présenterons dans un premier temps les principales plateformes parallèles et distribuées. Nous focaliserons notre étude sur les dernières technologies Hadoop et Spark pour leur efficacité à traîter de grandes volumétries de données. Nous présenterons également des plateformes exclusivement destinées aux graphes.

2.2.1 Hadoop

Apache Hadoop, est un logiciel open source, développé sous l'égide de la fondation Apache depuis 2005, dont la version 1.0 a vu le jour en 2011. Elle consiste en deux grandes composantes autour desquelles s'agrègent d'autres fonctionnalités. La première grande composante est le framework [1] MapReduce (Dean et Ghemawat (2008)), la seconde est le HDFS (pour Hadoop Distributed File System).

1. un framework ou structure logicielle est un ensemble de composants logiciels, qui sert à créer les fondations de tout ou d'une partie d'un logiciel.

MapReduce est un patron de conception destiné à effectuer des analyses et des opérations pour de grandes quantités de données et qui permet de distribuer les données sur plusieurs machines . Cette grappe de machines est communément appelée *cluster*. MapReduce comprend trois grandes étapes. Une *fonction Map* qui prend en paramètre un bloc de données du lecteur d'entrée et le traite. Cette fonction génère un ensemble de paires intermédiaires $< cle, valeur >$. Une *partie de shuffle* où l'ensemble des paires de $< cle, valeur >$ émis par les mappers est trié selon les clés, scindé en lots puis écrit sur le disque. Enfin, la *fonction Reduce* est invoquée pour chaque clé intermédiaire distincte et applique un traitement aux valeurs associées à ces mêmes clés. Le langage natif de programmation sous Hadoop est le Java, mais il a été étendu également aux langages Python et R.

Hadoop intègre un système de fichiers distribués qui prend en charge toutes les fonctions de réplication des données et de tolérance aux pannes sur un cluster. Il s'agit du HDFS. Le HDFS définit la notion de bloc qui correspond à la plus petite quantité de données qu'il est possible de lire ou d'écrire sur un disque. Cette taille est de 64 Mo. L'architecture du HDFS consiste en un système *maître-esclave*. Un nœud maître, appelé *NameNode* permet de gérer l'espace de nommage et les métadonnées du système de fichier. Il associe les blocs aux différentes machines du cluster. Des processus associés à chaque nœud, que l'on nomme *DataNode* gèrent les opérations de stockage locales, le tout dirigé par les instructions du NameNode. Par défaut, chaque fichier sur le HDFS est répliqué trois fois sur des machines différentes au sein du cluster, pour prévenir et assurer la tolérance aux pannes.

D'autres fonctionnalités s'agrègent autour du logiciel Hadoop pour répondre à des contraintes très spécifiques. Le tableau suivant récapitule les fonctionnalités annexes du logiciel Hadoop.

Les principales distributions d'Hadoop sont *Cloudera*, *Hortonworks*, *MapR* et *Amazon Elastic MapReduce*.

2.2.2 Apache Spark

Spark (Zaharia *et al.* (2010)) est un projet open source d'analyse de données ayant vu le jour en 2009 à Berkeley et qui fut l'un des projets prioritaires de la fondation Apache depuis février 2014. Spark utilise l'infrastructure distribuée de Hadoop, en particulier le HDFS, sans toutefois faire appel au patron de conception MapReduce. Les performances annoncées en termes de vitesse sont jusqu'à 100 fois plus importantes grâce à son architecture *in memory*. Le développement Spark se veut fonctionnel, c'est en ce sens que le langage de programmation natif de Spark est le Scala. Il est actuellement étendu aux langages Python et R.

L'un des atouts de Spark vis-à-vis d'Hadoop est de maintenir les résultats intermédiaires en mémoire plutôt que sur le disque, ce qui représente un avantage

Fonctionnalités agrégées autour de Hadoop	
HBase	Base de données NOSQL orientée colonne
Zookeeper	Service de coordination de processus distribués au moyen d'un espace de nommage partagé disponible en lecture
Pig	Langage procédural qui parse, optimise et exécute des scripts PigLatin comme une série de jobs MapReduce au sein d'un cluster Hadoop.
Hive	Infrastructure de datawarehouse construite sur Hadoop pour simplifier l'analyse d'ensembles de données très volumineuses.
Oozie	Outil permettant de construire des combinaisons complexes MapReduce, PigLatin ou Sqoop en une seule unité logique et permet la planification de jobs Hadoop.
Flume	Solution permettant d'agréger en temps réel différents flux de logs de serveurs webs.
Sqoop	Outil permettant de transférer efficacement de grands volumes de données entre Hadoop et des SGBD traditionnels comme MySQL, Oracle, IBM DB2 et Microsoft SQL.

Tableau 2.1 – Composants de l'éco-système Hadoop

en termes de temps d'exécution. Spark a été conçu pour travailler aussi bien en mémoire que sur le disque. Si les opérations ne tiennent plus en mémoire, Spark utilise alors l'espace disque.

Spark se base sur la notion de RDD (Resilient Distributed Datasets) qui sont des collections pouvant contenir tout type de données, qui distribuent les données sur le cluster tout en supportant des opérateurs parallèles. Les RDD permettent de réarranger les calculs et d'optimiser le traitement. Ils sont aussi tolérants aux pannes. Les RDD supportent deux types d'opérations, les *transformations* et les *actions*. Les transformations ne retournent pas de valeur seule, elles retournent seulement un nouveau RDD. Aucune évaluation n'est effectuée lorsque l'on fait appel à une fonction de transformation. Les actions évaluent et retournent une nouvelle valeur. A l'instant où une fonction d'action est appelée sur un objet RDD, toutes les requêtes de traitement des données sont calculées (qui correspondent aux transformations) et le résultat est retourné. C'est en ce sens que l'on qualifie Spark de système paresseux (lazy) dans la mesure où les transformations ne s'effectuent pas de suite.

Tout comme Hadoop, Spark contient son propre éco-système sur lequel s'agrègent différentes librairies et fonctionnalités. Nous récapitulons les librairies les plus importantes dans le tableau 2.3.

Fonctionnalités agrégées autour de Spark	
Spark Streaming	Utilisé pour le traitement en temps réel. En s'appuyant sur un mode de traitement de "micro batching", il utilise pour les données temps-réel DStream, c'est-à-dire une série de RDD.
Spark SQL	Outil agissant comme un ETL. Les jeux de données Spark sont mis en exergue via l'API JDBC. Spark SQL exécute des requêtes de type SQL en utilisant les outils BI et de visualisation traditionnels.
Spark MLlib	MLlib est une librairie contenant un grand nombre d'algorithmes et utilitaires d'apprentissage classiques, comme la classification, la régression, le clustering, le filtrage collaboratif, la réduction de dimensions, en plus des primitives d'optimisation sous−jacentes.
Spark GraphX	Outil pour le traitement des graphes avec des opérations parallèles dédiées. GraphX étend les RDD de Spark en introduisant le Resilient Distributed Dataset Graph (RDDG), un multi-graphe orienté avec des propriétés attachées aux nœuds et aux arêtes. On peut citer PageRank, la détection de composantes connexes, le parcours en largeur ou la propagation de labels comme algorithmes étant déjà implémentés sous cet outil.

Tableau 2.2 – Composants de l'éco-système Spark

2.2.3 Système de traitement de graphes parallèles

Il existe d'autres logiciels permettant la distribution et la parallélisation d'algorithmes comme OpenMP ou encore les systèmes MPI. Certains de ces logiciels sont intrinsèquement destinés aux graphes. Nous proposons de lister les principales solutions permettant le traitement de graphes et le développement d'algorithmes de graphes dans un environnement parallèle et distribué.

Pour gérer la parallélisation et la distribution d'algorithmes, les systèmes ont dû choisir entre différents modèles. Nous avons déjà vu le patron de conception MapReduce d'Hadoop ou encore la notion de RDD avec Spark, mais il en existe d'autres comme les BSP (pour "Bulk synchronous parallel") ou encore le modèle RAD (pour "Rassembler-Appliquer-Disperser").

Le modèle BSP (Valiant (1990)) décrit un ensemble de paires processeurs-mémoires homogènes en nombre fixe. De par un réseau de communication et une unité de synchronisation globale, les échanges de données entre proces-

seurs sont rendus possibles. L'exécution se déroule en plusieurs étapes, chacune constituée d'une phase de calculs locaux (effectuée sur chaque processeur) et d'une phase de communication entre les différents processeurs qui s'effectue avec des phases de synchronisation. Ce système garantit le déterminisme des applications développées et assure l'absence d'étreintes fatales (interblocages).

Le modèle RAD (Gonzalez *et al.* (2012)) est intrinsèquement destiné au graphe. Dans la première phase de "Rassemblement", un sommet accumule de l'information sur ses voisins. La seconde phase "Application" applique la valeur accumulée à ce sommet et met à jour ses sommets adjacents et arêtes. La dernière phase "Dispersion" active les sommets voisins. Les sommets peuvent directement tirer de l'information des sommets voisins sans toujours interagir avec eux, à l'instar du modèle BSP. Le modèle GAS permet des exécutions asynchrones sans l'aide de barrière de synchronisation.

A partir des deux modèles cités ci-dessus, des systèmes pour le traitement de graphes en environnement parallèle et distribué ont pu voir le jour.

Pregel (Malewicz *et al.* (2010)) est la première implémentation utilisant le principe du BSP. Ce logiciel a été orienté vers les sommets du graphe. Les calculs portant sur le graphe sont spécifiés en termes de ce que doivent faire les nœuds. Les arêtes servent de canaux et transmettent l'information résultante d'un sommet vers un autre mais ne participent pas au calcul. Un sommet peut avoir deux états : *actif* et *inactif*. Les nœuds actifs peuvent envoyer et recevoir les messages de leurs voisins (ou d'un autre sommet du graphe). Les super-étapes se terminent avec une barrière de synchronisation, assurant que tous les messages ont bien été transmis. Pregel se termine lorsque tous les sommets sont inactifs. Toutes les opérations avec Pregel sont effectuées en mémoire. Pregel ne permet pas l'écriture sur disque, ce qui oblige à avoir un important cluster de machines si la quantité de données à traiter est grande. L'architecture de Pregel est de type *maître-esclave*. Le nœud maître du cluster coupe les données en lots qui forment des sous-graphes. Chaque sous-graphe est envoyé à un "worker" spécifique (ordinateur du cluster qui effectue une tâche) qui travaille de manière indépendante des autres. Le nœud maître est responsable de la bonne synchronisation.

Giraph (Avery (2011)) est une solution open source construite sur Hadoop et utilisant le HDFS pour stocker les données. Giraph permet la création de structures de graphes et autorise certaines opérations MapReduce. Giraph utilise Zookeeper pour la coordination, les points de reprise et la défaillance des systèmes de récupération. Facebook (Ching (2013)) a utilisé Giraph avec quelques améliorations de performances pour analyser 1000 milliards d'arêtes (relations) en 4 minutes sur un cluster de 200 machines, dans le but de faire un algorithme de recherche d'information sur les sommets. Cependant, Giraph cessa depuis 2015 d'être documenté, cela étant dû principalement à la complexité de développement et au temps de traitement des données trop important pour certains algo-

rithmes, principalement globaux. De nombreux bugs ont également été signalés, décrédibilisant l'utilisation ce cette solution (Salihoglu *et al.* (2015)).

GPS (Salihoglu et Widom (2013)) (pour "A Graph Processing System") est un logiciel open source fondé sur Pregel. Des expérimentations ont montré qu'il était jusqu'à 12 fois plus rapide que Giraph sur certains algorithmes comme le parcours en largeur. Des optimisations par rapport à Pregel portent sur l'amélioration des communications, où les opérations se font par processeur dans un premier temps, puis entre processeurs dans un second temps, et non plus par voisinage comme sur Pregel. Cela permet une réduction de la synchronisation des threads (moins d'échanges entre processeurs) et une amélioration de la performance d'exécution en termes de temps. Une caractéristique de GPS est un partitionnement de graphe dynamique pour la répartition des données sur le cluster. Alors que certains systèmes partitionnent le graphe avant d'effectuer un processus, des migrations dynamiques de sommets ont lieu entre machines du cluster. Cette migration est fondée non sur une équirépartition des sommets (répartition de manière uniforme) du graphe au sein du cluster, mais sur une équirépartition des messages envoyés entre sommets afin de minimiser leur nombre. Cependant, la mise à jour des listes de voisins lors des migrations entraîne un gain de temps qui ne permet pas au système d'être plus performant en termes d'exécution que le modèle de base de Pregel. GPS offre aussi les "grandes listes d'adjacences partitionnables" (GLAP) qui permettent de gérer des nœuds avec beaucoup de connexions. Une migration des algorithmes de graphes s'effectue à ce jour vers GraphX.

Mizan (Khayyat *et al.* (2013); Kalnis *et al.* (2012)) est un logiciel open source fondé sur l'architecture de GPS avec des améliorations concernant les structures de données utilisées et les méthodes de partitionnement. Il a été reporté comme étant deux fois plus rapide que Giraph en mode statique (sans l'option de migration de nœuds au cours de la partition). Mizan a un partitionnement se fondant sur la topologie du graphe. L'algorithme, par une estimation de Kolmogorov (Pettitt et Stephens (1977)), peut détecter si le graphe est un graphe de terrain, auquel cas, un partitionnement spécifique aura lieu, dans lequel les hubs (nœuds avec de fortes connexions) seront répliqués sur plusieurs processeurs. Comme GPS, Mizan a une option avec partitionnement dynamique au cours de l'exécution d'un programme en se fondant sur la distribution des messages entre sommets du graphe. On reporte cependant de très nombreux bugs. La documentation laisse à penser que le projet n'est plus d'actualité.

GraphLab (Low *et al.* (2014)) est un projet open source qui intègre les fonctionnalités de Powergraph (Gonzalez *et al.* (2012)). GraphLab utilise le modèle RAD. L'une des différences avec les autres logiciels est que le partitionnement ne concerne que les nœuds, pas les arêtes. Les arêtes ne sont assignées que sur une seule machine alors que les nœuds sont répliqués dans la mémoire cache des machines du cluster suivant la distribution de leurs degrés (cela prévaut pour les hubs). Cela a pour conséquence un gain de temps notable. GraphLab supporte

les modes d'exécution synchrone et asynchrone. Les sommets qui n'envoient pas de messages ne sont pas considérés lors des calculs et retirés du cache, ce qui améliore le temps de calcul.

HAMA (Seo *et al.* (2010)) est un logiciel fondé sur le modèle BSP tout en étant construit sur Hadoop. Il n'est pas exclusivement destiné aux graphes mais présente de très nombreux avantages. Il fut conçu pour gérer de très grandes matrices ainsi que leurs opérations élémentaires (multiplication, inversion, etc.). Il est principalement destiné à l'algèbre linéaire. HAMA présente cependant un intérêt particulier pour les algorithmes itératifs car il permet de conserver les données en mémoire et de ne pas les écrire sur le disque, tel un précurseur de Spark. Les expérimentations réalisées sur PageRank ont montré que HAMA était jusqu'à trois fois plus rapide que Giraph.

Systèmes permettant le traitement de graphes parallèles				
Système	Langage	Modèle	optimisation	partitionnement
Hadoop	Java	MapReduce	-	hashage
Spark	Scala	RDD	réutilisation des données	1D−2D
Pregel	$C++$	BSP	-	hashage
GPS	Java	BSP	GLAP, migration dynamique	GLAP
Mizan	$C++$	BSP	migration, réplication	Mizan $\alpha - \beta$ hashage
Giraph	Java	BSP, MapReduce	écriture en mémoire	hashage
GraphLab	$C++$	RAD	calcul synchrone et asynchrone	Metis, ParMetis, hashage
Powergraph	$C++$	BSP,RAD	graphes de terrains avec une coupe minimale	sur les arêtes

Tableau 2.3 – Caractéristiques globales des systèmes pour le traitement de graphes de manière parallèle et distribuée

2.3 Détection de communautés dans un environnement parallèle et distribué

Des méthodes issues de la littérature de la détection de communautés ont été appliquées et retranscrites pour de grands graphes. Nous nous proposons d'exposer les plus importantes et celles qui permettront d'aboutir à une stratégie

pour notre solution concernant les grands graphes.

Riedy *et al.* (2011) ont implémenté l'algorithme de Clauset et al., algorithme agglomératif optimisant la modularité, en utilisant le multithreading Cray XMT. Cray XMT est une gamme de super-ordinateurs qui possèdent plusieurs processeurs. L'algorithme fut lancé sur un graphe de deux milliards d'arêtes et 122 millions de sommets avec plus de 128 processeurs. Les résultats sont satisfaisants mais la qualité ne fut jugée que par le score de la modularité. Les technologies Cray XMT permettent de réaliser des calculs sur de gros volumes de données mais sont très onéreuses. De plus, depuis l'apparition des technologies Hadoop et Spark, la technologie Cray XMT est de moins en moins utilisée. Riedy *et al.* (2012) ont amélioré leur algorithme en y incluant une implementation OpenMP. En utilisant 4 processeurs de 80 cœurs, les auteurs ont fait marcher leurs algorithmes sur un graphe de 3,3 milliards d'arêtes en 500 secondes. Cependant, les tests concernant la qualité ne purent être faits à cause de la taille des graphes.

Bahmani *et al.* (2012) ont proposé un algorithme pour détecter des groupes de nœuds denses, fondé sur la maximisation de fonctions fondées sur la densité de sous graphes. Soit $G = (V, E)$ et $S \subseteq V$, la densité de Bahman Bahmani et al. est définie par $\rho(S) = \frac{|E(S)|}{|S|}$. Plus $\rho(S)$ augmente, plus la structure se rapproche de celle d'un graphe complet. Les auteurs proposent un algorithme avec deux versions. La première concerne les graphes denses et construira la communauté S si l'inégalité $deg_S(i) \leq 2(1+\epsilon)\rho(S)$ est vérifiée, avec ϵ un réel donné par l'utilisateur. On peut comprendre cette inégalité comme le fait que chaque nœud doit avoir un degré suffisant pour appartenir à la communauté S. La seconde version, destinée à des graphes très denses, doit vérifier pour la création de structures communautaires l'inégalité suivante $deg_S(i) \leq \frac{\epsilon}{1+\epsilon}|S|$. Les deux versions opèrent de la même manière. Des nœuds sont enlevés dans le but de maximiser la densité des structures communautaires. L'inconvénient majeur de cette méthode est la paramétrisation qui a une grande influence sur la qualité de partitionnement. Il faut déjà avoir une idée de la densité ρ_G du graphe résultant que l'on cherche à obtenir mais également avoir à l'esprit une taille moyenne pour les communautés à ϵ près. La dernière contrainte peut poser problème pour des graphes complexes, où la taille des communautés peut être très variable. On peut en effet, avec une mauvaise paramétrisation, détecter des communautés contenant elles mêmes d'autres communautés. L'une des forces de cet algorithme est d'avoir été appliqué à de grands graphes, avec plus de 6 milliards d'arêtes avec 2000 reducers pour un temps d'exécution d'environ 260 minutes. La qualité de partitionnement des méthodes n'a pas été sujette à des études, le seul critère étant de maximiser les densités des communautés.

Tsironis *et al.* (2013) ont implémenté une méthode spectrale sous Hadoop. La méthode consiste à construire la matrice Laplacienne sur laquelle vont être calculés les K vecteurs propres associés aux K plus petites valeurs propres de la matrice, puis à lancer un k-means pour obtenir k clusters. Le calcul des vecteurs

propres se fait en utilisant HEIGEN (Kang *et al.* (2014)), un algorithme sous MapReduce reprenant la méthode de Lanczos.
Cette méthode présente l'inconvénient d'être fondée sur le k-means, mais est cependant destinée au partitionnement de graphes. De plus, il faut effectuer une recherche préalable pour sélectionner les vecteurs propres contenant le plus d'information. Bien que l'algorithme soit parallèle et distribué, la taille des graphes sur lesquels il a été appliqué est relativement faible, le plus grand graphe ne contenant que 20000 nœuds.

Ovelgonne (2013) propose un algorithme ensembliste fondé sur la propagation de label tout en réduisant la complexité du problème, via une réduction de la taille des données de départ. En lançant en parallèle plusieurs propagations de labels, une méthode de chevauchement maximale permet de trouver les cœurs. Cependant, Ovelgönne n'a pas appliqué la méthode asynchrone sous Hadoop car trop gourmande en temps. Il choisit plutôt une version avec mise à jour en fonction de probabilités de telle sorte que le nombre de propagations de labels synchrones soit minimisé. Cela détériore la qualité de partitionnement. De plus, pour éviter d'écrire de trop nombreuses fois sur le disque, certains nœuds ne sont mis à jour qu'une seule fois avec un nombre d'itérations assez faible dès le départ.
Les résultats ont été appliqués sur de très grands réseaux de terrain comme LiveJournal avec une heure pour 20 propagations de labels en parallèle sur un cluster de 50 machines. Les résultats sont évalués avec la modularité qui donne des valeurs correctes pour la qualité de partitionnement.

Staudt et Meyerhenke (2013) proposent une version "OpenMP" implémentant la propagation de labels et la méthode de Louvain en $C++$, avec trois méthodes fondées sur le clustering d'ensemble et vote par consensus :

— Ensemble de prétraitement
— Ensemble multi-niveaux
— Clustering de cœur et contraction de graphe

Ensemble de prétraitement (EPP) : il consiste dans un premier temps à assigner le graphe à plusieurs algorithmes de détection de communautés. Une contraction du graphe a lieu selon un consensus entre tous les résultats des différentes détections de communautés. Le graphe est ainsi contracté. L'objectif de ce système est de pouvoir allier qualité (par le consensus) et rapidité (par la contraction).
Ensemble multi-niveaux (EM) : il est une amélioration de l'EPP dans un sens itératif. A la fin de l'étape de contraction, un nouveau graphe G' est créé et le processus fondé sur l'EPP continue de manière itérative. Les conditions pour arrêter la contraction sont multiples comme i) obtenir un singleton ii) mettre un seuil sur la taille du graphe réduit en dessous duquel la contraction est interdite iii) la présence d'une diminution de la modularité d'une contraction à la suivante et iv) la stagnation de la qualité des communautés détectées.
Clustering de cœur et contraction de graphe (CCGC) : représente la méthode la plus stricte en termes de consensus. Si une paire de nœuds a

été identifiée comme appartenant à la même communauté, et ceci dans toutes les détections de communautés, alors ces deux nœuds seront contractés. Cette dernière méthode permet de trouver les meilleurs cœurs en termes de qualité. Cependant, il se peut que certains résultats de détection de communautés ne soient pas de bonne qualité. La conséquence est que des cœurs ne puissent être trouvés.
C'est en utilisant ces dernières méthodes que la propagation de labels pondérée et la méthode de Louvain ont été appliquées dans leurs versions parallèles.
Le parallélisme de la propagation de labels sur OpenMP (noté PLP) réside en un tri des nœuds se faisant sur plusieurs threads qui opérent sur des tableaux de labels communs. La mise à jours des labels peut ainsi se faire de manière synchrone ou asynchrone. Il s'agit d'une implémentation pour des graphes pondérés, avec une mise à jour des labels dont les arêtes liant le nœud sont les plus importantes. Un seuil δ doit cependant être spécifié en dessous duquel le changement de labels devient impossible.
Le parallélisme de l'algorithme de Louvain sur OpenMP intervient dans le calcul de la fonction de modularité, qui se fait en utilisant plusieurs threads. En effet, chaque mouvement d'un nœud dans une autre communauté a pour conséquence la variation de la modularité qui est calculée localement.
Il ressort des expérimentations que CCGC semble donner les meilleurs résultats en termes de qualité pour PLM et PLP. PLP est extrêmement rapide mais de qualité moyenne, alors que PLM obtient de meilleurs résultats, mais nécessite plus de temps.

Moon *et al.* (2014) développèrent l'algorithme de Girvan et Newman (2002b) sous Hadoop. Une version du plus court chemin fut développée sous MapReduce en quatre étapes. La première phase consiste à calculer entre chaque paire de nœuds le plus court chemin. La seconde étape consiste dans le calcul de la centralité d'intermédiarité. La troisième étape permet de sélectionner k arêtes ayant les centralités d'intermédiarité les plus élevées. k est un paramètre de l'utilisateur. La dernière phase consiste à retirer les arêtes sélectionnées. L'algorithme fut appliqué à deux graphes de petite taille de 6000 et 10000 nœuds avec respectivement 290 000 et 51 000 arêtes. Avec plus de 20 machines, le système requiert un temps d'exécution de près de 300 secondes. Bien que les résultats en termes de qualité soient encourageants, l'algorithme présente deux inconvénients majeurs. Le premier est le choix du nombre d'arêtes à retirer qui n'est *a priori* pas connu de l'utilisateur. Le second inconvénient est le temps de calcul du plus court chemin entre chaque paire de nœuds qui est beaucoup trop important pour des graphes ayant quelques dizaines de milliers d'arêtes. De ce fait, l'algorithme ne peut pas être appliqué à de grands graphes de plusieurs millions d'arêtes.

Kuzmin *et al.* (2015) ont implémenté le SLPA dans sa version parallèle dans un contexte multithreading. Chaque thread réalise une propagation de labels sur un sous ensemble de nœuds qui a été préalablement partitionné. Le partitionnement est utilisé pour ne pas dépasser la mémoire vive. C'est alors que les

sous-graphes résultants sont assignés à des threads où les propagations de labels sont lancées.
Les résultats ont montré que la méthode était équivalente en termes de qualité de partionnement à la version non parallèle. Elle obtient cependant de meilleurs résultats en choisissant une méthode de partitionnement particulière, mais une étude préalable doit être effectuée. Elle fut lancée sur des graphes comme DBLP, Amazon ou Livejournal, avec plusieurs millions de sommets et jusqu'à une centaine de millions d'arêtes.

La propagation de label a également été développée sous Apache Spark, par une implémentation et par une intégration Pregel dont le modèle suit un BSP. Comme nous l'avons vu, Pregel est un BSP qui permet l'implémentation d'algorithmes sur les graphes. Pregel fut construit sur la philosophie "Penser comme un sommet". Chaque calcul est fait sur chaque sommet et les arêtes sont des canaux de communication permettant d'envoyer ou de recevoir des informations ou opérations des voisinages des nœuds (ou même d'un nœud à un autre nœud en connaissant l'identifiant). Pregel finit une "super étape" lorsque tous les nœuds du graphe sont dans un état inactif. La propagation de labels fut ainsi implémentée de manière synchrone où chaque itération consiste en une "super étape". A la fin d'une super étape, chaque nœud prend connaissance du label majoritaire de son voisin et un vote est ainsi effectué. Le processus se continue un nombre de fois spécifié par l'utilisateur. En considérant l'implémentation sur Spark, à chaque super étape, un `RDD` (Resilient distributed dataset) est créé et est utilisé pour l'itération de la propagation de labels suivante. La signature du LPA sous Spark est `lib.LabelPropagation.run(`par_1`, `par_2`).vertices` où par_1 se réfère au graphe où sera appliqué le LPA et par_2 sera le nombre d'itérations (de super étapes) du LPA. Cette implémentation peut être appliquée à des graphes de plusieurs millions d'arêtes en utilisant un cluster de machines important, mais souffre d'un inconvénient qui lui est intrinsèque. Son implémentation synchrone laisse l'algorithme instable et non détermiste. La solution pour résoudre ce problème est que l'ordre de visite des nœuds soit toujours le même, afin de rendre plus déterministe l'algorithme. Cependant, si l'ordre de visite commence par un nœud qui est situé entre plusieurs structures communautaires, de mauvaises propagations peuvent se produire.

2.4 Tableau récapitulatif des méthodes parallèles et distribuées

Classification des principaux algorithmes de détection de communautés parallèles et distribués					
Algorithmes parallèles	articles	d	c	complexité (ordre)	framework
Zhang et al	(Zhang *et al.*, 2009)	non	non		Hadoop
Bahman et al.	(Bahmani *et al.*, 2012)	non	non		Hadoop
EJ Riedy et al.	(Riedy *et al.*, 2011)	non	non		Cray XMT
EJ Riedy et al.	(Riedy *et al.*, 2012)	non	non		Cray XMT, OpenMPI
Tsironis et al.	(Tsironis *et al.*, 2013)	non	non	$\mathcal{O}(exp(n))$	Hadoop
Ovelgonne	(Ovelgonne, 2013)	non	non	$\mathcal{O}(kmn)$	Hadoop
Staudt et al.	(Staudt et Meyerhenke, 2013)	oui	non	$\mathcal{O}(kmn)$	MPI
Moon et al.	(Moon *et al.*, 2014)	non	non	$\mathcal{O}(kmn)$	Hadoop
A Prat-Pérez	(Saltz *et al.*, 2015)	non	non	$\mathcal{O}(kmn)$	Giraph-Hadoop
Kuzmin et al.	(Kuzmin *et al.*, 2015)	non	oui	$\mathcal{O}(kmn)$	MPI

Tableau 2.4 – Principaux algorithmes de détection de communautés parallèles et distribués. "*" dépend du paramétrage, difficile à estimer, d pour dendrogramme et c pour chevauchement. Première partie.

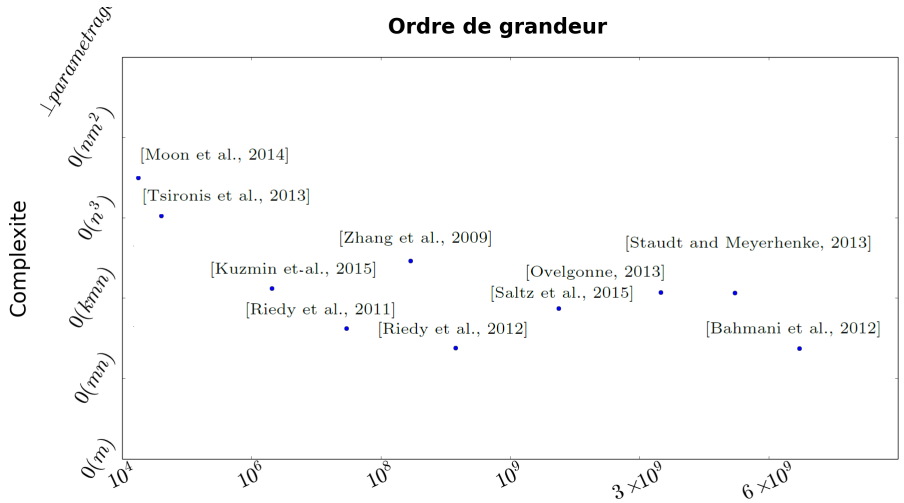


Figure 2.1 – Méthodes parallèles et distribuées. L'axe des abscisses représente la taille des graphes, l'axe des ordonnées, la complexité.

En étudiant la figure 2.1, on voit que les algorithmes permettant de traiter les plus grands graphes concernent les méthodes portant sur la maximisation de la densité des structures communautaires et celles se fondant sur la propagation de labels. Nous observons que les méthodes gobales telles que la méthode spectrale ou celle concernant la centralité d'intermédiarité ne permettent pas de travailler avec des graphes de plus de 30 000 nœuds. Cependant, ces dernières représentent une avancée sur la taille des réseaux vis-à-vis de leurs homologues originellement non parallèles et non distribuées.

2.5 Synthèse et discussion

Ce chapitre nous a permis d'exposer la problématique de la détection de communautés dans de grands graphes dont les difficultés majeures résident dans :

- le stockage des données massives du fichier représentant le graphe et également durant le processus de l'algorithme.
- l'échelonnabilité, c'est-à-dire le fait qu'un algorithme puisse être capable de travailler à la fois sur de petits graphes et de grands graphes en donnant la même qualité de partitionnement et dont le stockage de données soit linéaire.

Ce seront les challenges pour notre contribution sur de grands graphes.

Ce chapitre a également permis de voir les plus importantes plateformes parallèles et distribuées existantes, particulièrement les PGPS destinés aux graphes. C'est en ce sens que nos choix se porteront principalement sur deux plateformes Hadoop et Spark. Les raisons principales sont leur faible coût financier et le fait que de nombreux algorithmes de graphes furent développés en utilisant ces plateformes, comme PEGASUS (Kang *et al.* (2009)) (le degré, PageRank, marche aléatoire avec redémarrage (MAR), le diamètre, les composantes connexes, etc.) et Giraph pour Hadoop et GraphX (Avery (2011)) pour Spark (Zaharia *et al.* (2010)). Hadoop permet l'utilisation du HDFS qui autorise le traitement de données massives et la tolérance aux pannes. Cependant, cette plateforme connaît des difficultés dans le traitement de données en temps réel, ce qui peut être le cas de la propagation de labels où tout autre nœud doit connaître le label de ses voisins en temps réel (version asynchrone). C'est un challenge auquel nous souhaitons faire face. L'état de l'art nous a permis de voir que les méthodes à propagation de labels permettaient de traiter de grands graphes, c'est donc l'option que nous privilégions.

Chapitre 3

Propositions algorithmiques sur la détection de communautés

Sommaire

L'état de l'art concernant la détection de communautés disjointes nous a permis de voir l'existence de trois grandes classes d'algorithmes, à savoir les méthodes globales, les méthodes locales et les méthodes hybrides. Dans le but de créer un algorithme pouvant travailler sur de grands graphes, nous avons privilégié d'utiliser la propagation de labels. Cependant, l'algorithme de propagation de labels souffre de certains problèmes liés à des choix aléatoires, comme celui du choix d'un label lorsqu'il y a équidistribution des labels majoritaires au sein du voisinage d'un nœud. Cela peut mener à :

— de mauvaises propagations de labels (susceptibles de donner des communautés géantes, groupes de nœuds où les structures n'ont pu être détectées) ou des communautés non connectées avec le même label)
— l'instabilité (ne donnant que très rarement la même partition après plusieurs lancements)
— l'impossibilité de créer un dendrogramme
— l'impossibilité de trouver des communautés chevauchantes.

Dans cette section, nous allons exposer nos propositions hybrides, fondées sur la propagation de labels en utilisant la détection de cœurs et la mise en place de barrages artificiels sur certaines arêtes interdisant toute propagation. Nous ferons une étude portant sur la qualité de la détection de nos algorithmes mais aussi sur leurs paramétrages. Par la suite, nous proposerons des algorithmes, fondés sur nos méthodes disjointes, ayant pour vocation le chevauchement. L'objectif de la partie de chevauchement est de créer une méthode permettant à un nœud d'appartenir à autant de communautés que nécessaire suivant certaines conditions que nous expliciterons. Une étude comparative sera effectuée.

Dans la suite de ce manuscrit, nous notons par $s_u \leftarrow s_v$ le fait que la valeur du label du nœud u prenne la valeur du label du nœud v.

3.1 Propagation de labels avec détection de cœurs, barrages et dendrogrammes

Pour diminuer la probabilité qu'une mauvaise propagation puisse survenir, nous avons réfléchi à un moyen de limiter cette dernière. Il serait souhaitable que la propagation de labels puisse se faire dans des régions du graphe densément connectées en évitant de mauvaises propagations entre régions faiblement connectées, ce qui aurait pour conséquence d'obtenir de trop grosses communautés.

Considérons deux exemples donnant une mauvaise propagation de labels. Le

premier traite des communautés géantes alors que le second traite de communautés non liées ayant les mêmes labels.

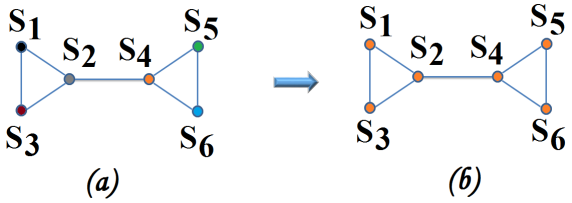


Figure 3.1 – Exemple d'une mauvaise propagation donnant une communauté géante

En considérant l'odre de visite sur le graphe de la figure 3.1

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ s_2 & s_5 & s_6 & s_4 & s_1 & s_3 \end{pmatrix}$$

Une suite d'opérations possibles peut être $s_2 \leftarrow s_4$ (par choix aléatoire), $s_5 \leftarrow s_4$ (par choix aléatoire) $s_6 \leftarrow \{s_4, s_5\}$ (par vote majoritaire) $s_4 \leftarrow \{s_2, s_5, s_6\}$ (par vote majoritaire) $s_1 \leftarrow s_2$ (par choix aléatoire) et $s_3 \leftarrow \{s_2, s_1\}$ (par vote majoritaire). Ainsi, les deux structures communautaires, à savoir $\{s_1, s_2, s_3\}$ et $\{s_5, s_4, s_6\}$ n'ont pas pu être détectées. Il s'agit d'une communauté géante.

Un autre exemple porte sur des communautés ayant un même label mais qui ne sont pas reliées.

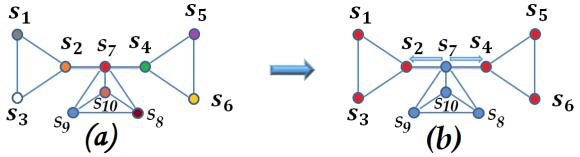


Figure 3.2 – Exemple d'une mauvaise propagation donnant des communautés avec le même label. s_7 a transmis son label à deux structures communautaires

Opérations de la figure 3.2 : $s_2 \leftarrow s_7$, $s_4 \leftarrow s_7$,$s_5 \leftarrow s_4$,$s_6 \leftarrow \{s_4, s_5\}$, $s_1 \leftarrow s_2$, $s_3 \leftarrow \{s_2, s_1\}$, $s_{10} \leftarrow s_9$, $s_8 \leftarrow \{s_9, s_{10}\}$, $s_9 \leftarrow \{s_8, s_{10}\}$ and $s_7 \leftarrow \{s_8, s_9, s_{10}\}$.

L'idée est donc d'imaginer une méthode pour interdire ce genre de mauvaises propagations.

3.1.1 Propagation de labels asynchrone avec barrages

La solution que nous proposons est de mettre des barrages artificiels permettant de stopper certaines propagations, particulièrement sur des liens où le risque de mauvaises propagations est élevé, ce qui pourrait induire par la suite la création de trop grandes communautés.

La propagation de labels avec barrages a pour objectif de détecter des communautés, en utilisant à la fois l'information globale topologique du graphe et une méthode locale. La propagation de labels souffrant d'instabilité et du fait

de produire de trop grandes communautés, nous proposons une méthode pour résoudre ces deux problèmes en interdisant à certaines arêtes de propager un label tout en effectuant une détection de cœurs en lançant plusieurs fois l'algorithme non déterministe. Pour une raison de stabilisation (Raghavan *et al.* (2007)), nous privilégions la version asynchrone de la propagation de labels.

Les mesures issues de l'analyse des réseaux sociaux peuvent se révéler très utiles pour connaître l'importance de certaines arêtes ou sommets au sein d'un graphe. Nous proposons d'utiliser la mesure d'intermédiarité de Girvan et Newman (2002a) (présentée au premier chapitre) qui nous permettra de placer des barrages lors de l'exécution de la propagation de labels. Ce choix vient du fait que cette mesure permet de déceler des arêtes connectant des groupes de nœuds généralement densément connectés.

Nous proposons d'utiliser la centralité d'intermédiarité pour mettre nos barrages et éviter l'obtention de trop grandes communautés. Nous notons cet algorithme, la propagation de labels avec barrages (PLAB) (Algorithme 2). La complexité de la centralité d'intermédiarité étant en $\mathcal{O}(n^3)$ (Fortunato (2010)), la complexité de PLAB est en $\mathcal{O}(n^3 + k(m - \beta \times m))$, où k est le nombre d'itérations pour les différentes propagations de labels.

Algorithme 2 PLAB

Paramètres : Un graphe $G = (V, E)$, un réel β (pourcentage de barrages)
Sortie : Une partition $P = \{P_1, ..., P_\mathcal{C}\}$ (communautés de G)

1: Calcul de la centralité d'intermédiarité de G
2: Sélectionner β pourcentage des arêtes ayant les plus grandes valeurs de centralité d'intermédiarité et mettre des barrages
3: Lancer la propagation de labels asynchrone.
4: **Retourner** Une partition $P = \{P_1, ..., P_\mathcal{C}\}$

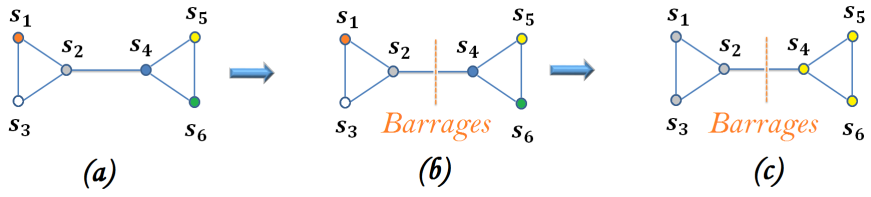


Figure 3.3 – Exemples de propagation de labels avec barrages a) le graphe originel b) mise en place d'un barrage sur l'arête avec la plus forte centralité d'intermédiarité c) lancement de la propagation de labels

Dans l'exemple de la figure 3.3, le graphe comprend 6 sommets. On calcule dans un premier temps la centralité d'intermédiarité des arêtes. Un tri est alors effectué sur les arêtes. Une sélection permet de prendre l'arête ayant la plus

grande valeur d'intermédiarité. On place alors un barrage artificiel qui sera utilisé lors de la propagation de labels. Ce barrage interdit à la propagation de s'effectuer. On trouve par la suite deux communautés.

Notre première étude consiste à apprécier la mise en place de barrages afin d'observer une possible amélioration de la qualité par rapport à la propagation de labels standards. Nous proposons de faire varier le nombre de barrages sur certains réseaux classiques de la littérature dont nous connaissons les vraies structures communautaires, tout en calculant les mesures supervisées et non supervisées.

En utilisant PLAB sur le réseau de karaté, la Figure 3.4, l'ajout de barrages n'améliore pas significativement la qualité de partitionnement. La qualité se détériore par la suite avec une augmentation du nombre de communautés. Cependant, les résultats ne sont pas stables. On peut voir l'exemple sur le NMI, la Figure 3.4c, la présence de nombreux pics respectivement à 20%, 40% et 60%. Cela est dû à l'instabilité de la détection de communautés qui ne produit que très rarement le même résultat d'un test à un autre. On peut ainsi ne pas obtenir une décroissance linéaire, mais avec cassures.

En considérant l'exemple des clubs de foot où les communautés réprésentent des conférences, sur la Figure 3.5, on voit que l'apport de barrages augmente sensiblement la qualité de partitionnement. De $2,5\%$ à 55% de barrages, les résultats sont meilleurs que ceux du LPA, avec un NMI atteignant 0.93 pour PLAB par rapport à 0.88 pour le LPA. Des pics récurrents apparaissent, signalant l'instabilité de la méthode.

3.1.2 Propagation de labels asynchrone avec barrages et détection de cœurs

L'algorithme PLAB présente deux inconvénients : le choix du nombre de barrages, et l'instabilité réduite, mais toujours présente. Pour remédier à ces problèmes nous proposons deux algorithmes fondés sur la détection de cœurs. Le premier algorithme est fondé sur l'optimisation d'une fonction de qualité qui peut par exemple être la modularité ou la conductance. Nous effectuons plusieurs fois PLAB avec un certain β (le pourcentage de barrages sur les arêtes ayant la plus forte valeur de centralité d'intermédiarité) pour alimenter une matrice de fréquence. Cette matrice permet de voir, pour chaque paire de nœuds i et j, le nombre de fois qu'ils sont dans la même communauté au cours des $\mathcal{N}$ propagations de labels. Nous recommençons le procédé en changeant la valeur de β, β variant de 0 à 1 par un pas Δ choisi par l'utilisateur avec une nouvelle matrice de fréquence de fréquence. Pour chaque matrice de fréquence, nous calculons les composantes connexes qui représentent les communautés et effectuons une évaluation avec une fonction de qualité et en utilisant un seuil α. Le seuil α est un réel positif permettant de créer un nouveau graphe à partir d'une matrice

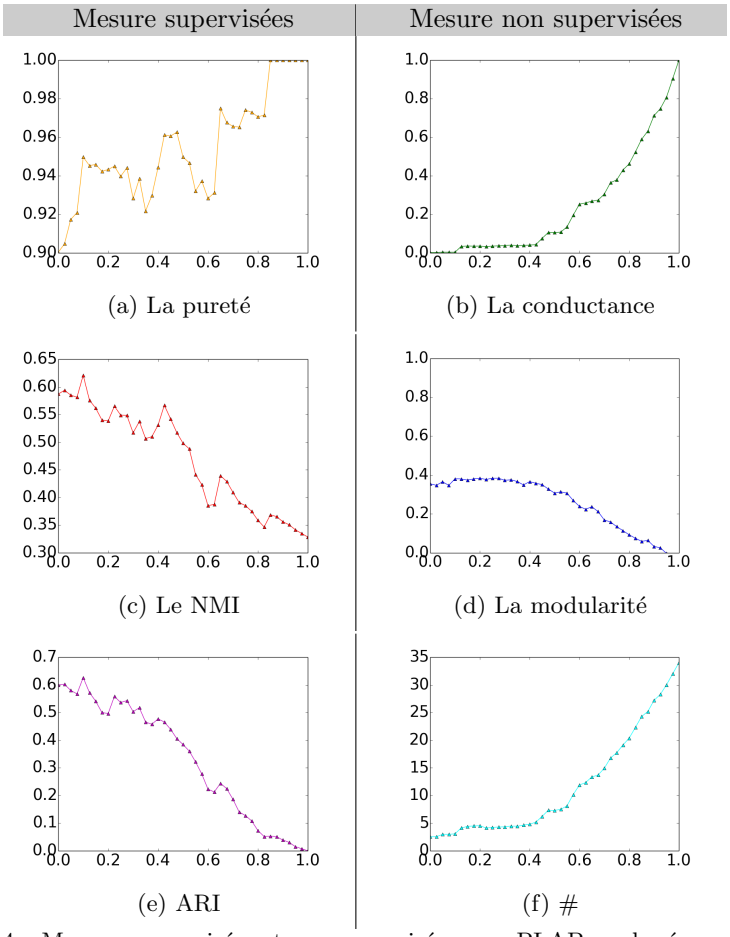
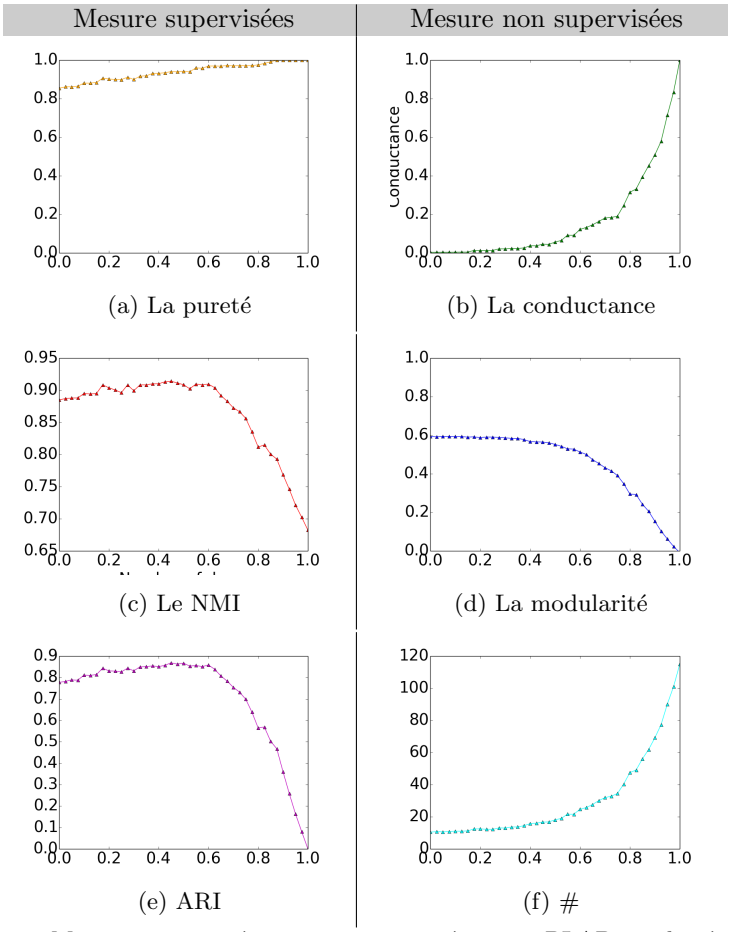


Figure 3.4 – Mesures supervisées et non supervisées avec PLAB sur le réseau Zachary. # représente le nombre de communautés. L'axe des abcisses représente le pourcentage de barrages.

de fréquence. En faisant varier α de 0 à 1 par un certain pas Δ, il est possible d'obtenir un dendrogramme. Chaque matrice de fréquence donne un dendrogramme en faisant varier le seuil α. Parmi tous les dendrogrammes, le résultat ayant le score optimal de modularité ou de conductance permet de retourner une partition. La complexité de l'algorithme est en $\mathcal{O}(n^3 + \frac{1}{\Delta} \times k \times \mathcal{N} \times (m - \beta m))$, où k est le nombre d'itérations pour les différentes propagations de labels.
Nous exposons ci-dessous la multiple propagations de labels avec barrages et stabilisation avec optimisation d'une fonction de qualité (MPLBS) (Algorithme 3).



(a) La pureté (b) La conductance

(c) Le NMI (d) La modularité

(e) ARI (f) #

Figure 3.5 – Mesures supervisées et non supervisées avec PLAB sur le réseau footballistique. # représente le nombre de communautés. L'axe des abcisses représente le pourcentage de barrages.

Nous donnons un exemple d'application sur un petit graphe de la méthode MPLBS, Figure 3.6. Dans cet exemple, chaque matrice est alimentée par différentes propagations de labels (ici 10 pour l'exemple) avec un certain pourcentage de barrages. On commence avec un barrage en appliquant 10 propagations de labels qui alimentent une matrice de fréquence, puis on alimente une nouvelle matrice de fréquence avec deux barrages et ainsi de suite jusqu'au nombre maximal de barrages possibles. On s'aperçoit que plus le nombre de barrages augmente, plus la matrice de fréquence se transforme en matrice diagonale. L'obtention d'une matrice diagonale signifie que chaque nœud représente sa propre communauté. En utilisant un seuil α que l'on fait varier entre 0 et 1, nous pouvons obtenir différents dendrogrammes selon les valeurs des éléments de la matrice de fréquence. Nous voyons que certaines matrices de fréquences sont identiques

Algorithme 3 MPLBS

Paramètres : Un graphe $G = (V, E)$, un seuil α, $\mathcal{N}$ le nombre de propagations de labels, Δ le pas

Sortie : communautés de G

1: Calcul de la centralité d'intermédiarité de G
2: $Liste\_Partition = []$
3: **Pour** $j = 0$ à 1 avec un pas de Δ **Faire**
4: Mettre des barrages sur les $j \times |E|$ arêtes ayant les plus grandes valeurs d'intermédiarité
5: Allouer une matrice vide $P_{ij}^{\mathcal{N}}(j)$
6: Alimenter la matrice $P_{ij}^{\mathcal{N}}(j)$ en lançant $\mathcal{N}$ fois la propagation de labels sur le graphe avec barrages.
7: Créer un nouveau graphe $G' = (V, E')$ en partant de $P_{ij}^{\mathcal{N}}(j)$ avec des arêtes dont la pondération est égale ou supérieure à α
8: Créer une partition P_j en considérant les $\mathcal{C}$ composantes connexes.
9: $Liste\_Partition.ajouter(P_j)$
10: $j = j + \Delta$
11: **Fin Pour**
12: **Retourner** la partition P_k de $Liste\_Partition$ avec le meilleur score de la fonction de qualité choisie par l'utilisateur : $P_k^* = \{P_1, ..., P_{\mathcal{C}}\}$

avec un nombre différent de barrages. Pour sélectionner le dendrogramme pouvant donner le meilleur résultat en termes de qualité de partitionnement, nous utilisons les mesures non supervisées de modularité et de conductance.

Le second algorithme fait varier le nombre de barrages en alimentant une seule matrice de fréquence, puis détecte les composantes connexes. L'idée est qu'alimenter une matrice de fréquence par une même méthode, qui ne produit pas toujours de bons résultats ne peut pas être satisfaisant. L'alimentation est faite en lançant la méthode non déterministe séquentiellement un nombre $\mathcal{N}$ de fois pour remplir la matrice de fréquence, et ainsi voir les nœuds ayant une forte probabilité d'être dans une même communauté. Cependant, alimenter une matrice, avec différentes méthodes, ayant différents niveaux de barrage pourrait améliorer la qualité des composantes connexes trouvées pour la détection de communauté. L'algorithme prend en paramètre, outre le seuil α et le nombre d'essais $\mathcal{N}$, l'intervalle considéré pour l'alimentation de notre matrice, $\Delta_{[x;y]}$, $x \in [0, 1]$,$y \in [0, 1]$ et $x < y$, avec un pas pour parcourir cet intervalle Δ. Nous exposons ci-dessous l'algorithme de propagation de labels avec barrages utilisant la stabilisation (PLBS) (Algorithme 4). La complexité de l'algorithme est en $\mathcal{O}(n^3 + k \times \mathcal{N} \times (m - \beta m) \times \frac{(y-x)}{\Delta})$, où k est le nombre d'itérations pour les différentes propagations de labels. Nous exposons un exemple d'application de la méthode PLBS, Figure 3.7. Dans cet exemple, une seule matrice de fréquence est alimentée par différentes propagations de labels avec des pourcentages de barrages différents. Nous observons que nous obtenons une matrice

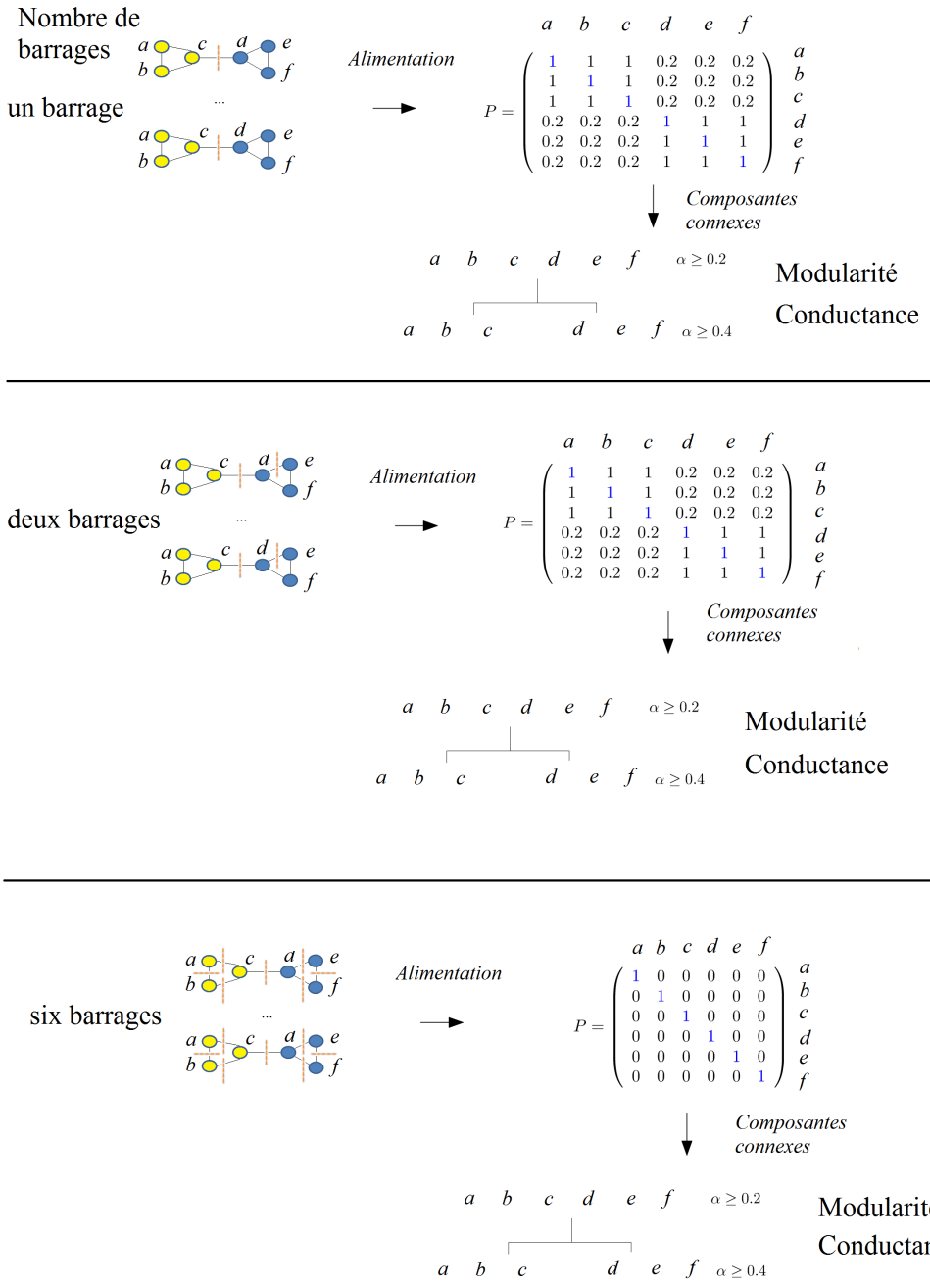


Figure 3.6 – Exemple de la multiple propagations de labels avec barrages et stabilisation avec optimisation d'une fonction de qualité

par bloc. Bien que le graphe possède une structure de symétrie (deux triangles), on peut observer que les valeurs des éléments dans la matrice de fréquence sont différents, notamment en comparant les triangles a, b, c et d, e, f. Cela vient du fait que lorsqu'il existe une symétrie, le choix pour mettre un barrage est aléatoire. Ainsi, il est possible que des barrages se mettent dans une structure communautaire alors qu'une autre structure, identique, attendra plus tard pour la mise en place de barrages en son sein. Par alimentation, les valeurs dans ces structures symétriques sont différentes, ce qui explique que la fréquence d'appartition des nœuds e et d soit de 0.4, différente de la fréquence d'apparition au sein d'une même communauté des nœuds a et c, qui est de 0.3. En utilisiant un seuil α, que l'on fait varier entre 0 et 1, un dendrogramme apparaît. Nous utilisons les mesures non supervisées de modularité et de conductance pour retourner la partition selon la valeur de α.

Algorithme 4 PLBS

Paramètres : Un graphe $G = (V, E)$, un seuil α, $\mathcal{N}$ le nombre de propagations de labels, Δ le pas, l'intervalle pour l'alimentation $\Delta_{[x;y]}$
Sortie : communautés de G
Calcul de la centralité d'intermédiarité de G
Allocation d'une matrice de fréquence vide
Pour $j = x$ à y avec un pas Δ **Faire**
  Mettre des barrages sur les $j \times |E|$ arêtes ayant les plus grandes valeurs d'intermédiarité
  Lancer $\mathcal{N}$ fois la propagation de labels avec un nombre différent de barrages
  Remplir la matrice de fréquence avec les résultats des différentes propagations de labels
  $j = j + \Delta$
Fin Pour
Créer un nouveau graphe $G' = (V, E')$ en partant de $P_{ij}^{\mathcal{N}}$ avec des arêtes dont la pondération est égale ou supérieure à α
Créer une partition P en considérant les $\mathcal{C}$ composantes connexes.
Retourner La partition $P = \{P_1, ..., P_{\mathcal{C}}\}$.

Enfin, nous proposons la propagation de labels avec détection de cœurs sans barrage, notée CDLP. La complexité de l'algorithme est en $\mathcal{O}(\mathcal{N} \times k \times (n+m))$, où k est le nombre d'itérations pour les différentes propagations de labels. Nous donnons un exemple de la propagation de labels avec détection de cœurs à la Figure 3.8. Dans cet exemple, une seule matrice de fréquence est alimentée par différentes propagations de labels. On voit que la matrice de fréquence donne une matrice par bloc, avec des valeurs différentes entre certaines paires de nœuds permettant de voir les deux structures communautaires. En faisant varier α de 0 à 1 pour obtenir un dendrogramme, on s'aperçoit que des valeurs différentes de α donnent parfois de mêmes partitions. Nous utilisons les mesures non su-

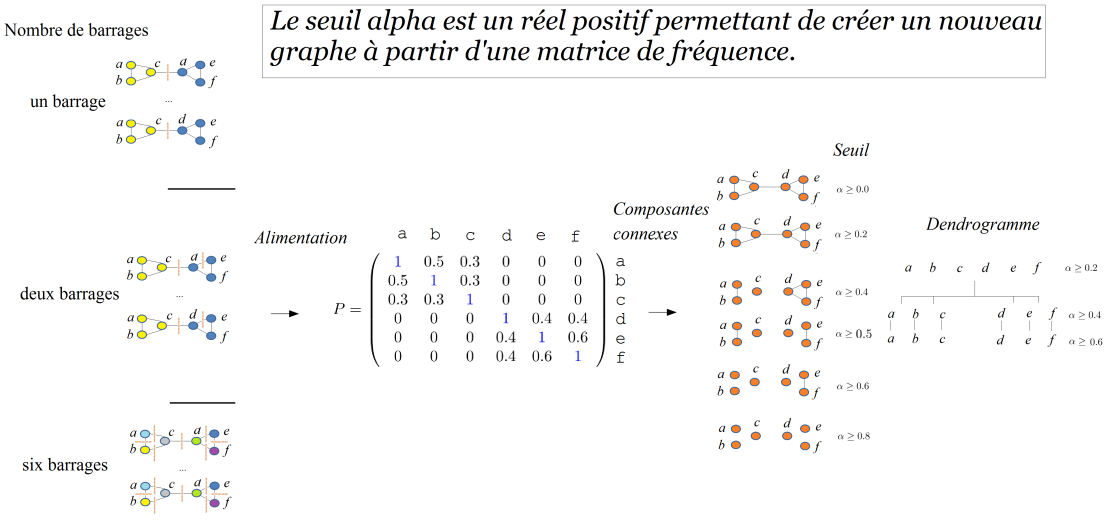


Figure 3.7 – Exemple de la propagation de labels avec barrages utilisant la stabilisation

pervisées de modularité et de conductance pour retourner la partition selon la valeur de α. L'objectif d'utiliser le CDLP dans nos futures expérimentations est d'observer si des barrages peuvent donner de meilleurs résultats en termes de qualité de partitionnement.

Algorithme 5 CDLP

Paramètres : Un graphe $G = (V, E)$, un seuil α, $\mathcal{N}$ le nombre de propagations de labels

Sortie : communautés de G

Allocation d'une matrice de fréquence vide

2: Lancer $\mathcal{N}$ fois la propagation de labels asynchrone.
Remplir la matrice de fréquence avec les résultats des différentes propagations de labels.

4: Créer un nouveau graphe $G' = (V, E')$ en partant de $P_{ij}^{\mathcal{N}}$ avec des arêtes dont la pondération est égale ou supérieure à α
Creér une partition P en considérant les $\mathcal{C}$ composantes connexes.

6: **Retourner** la partition $P = \{P_1, ..., P_{\mathcal{C}}\}$.

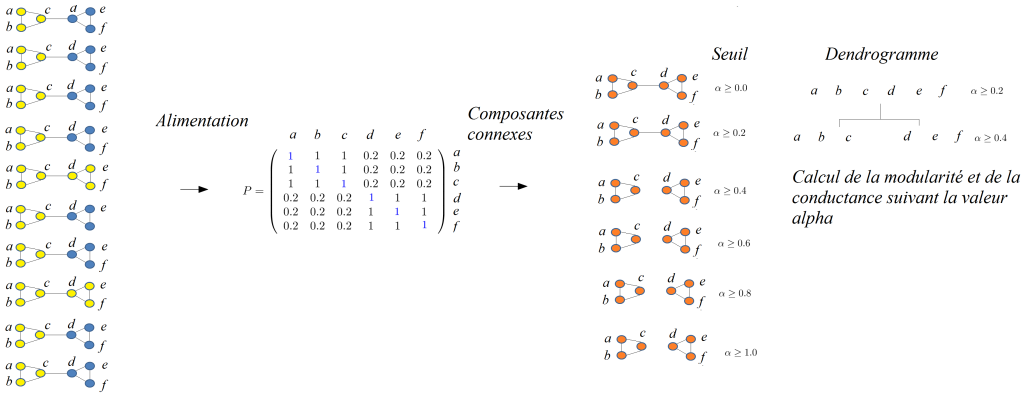


Figure 3.8 – Exemple de la propagation de labels avec détection de cœurs

3.1.3 Expérimentations portant sur les propositions algorithmiques pour la détection de communautés disjointes

Nous nous proposons d'effectuer notre étude sur des réseaux réels modélisant des phénomènes sociologiques, pour lesquels des experts ont établi les communautés qu'ils jugent correctes. Cependant, certains réseaux réels n'ont pas bénéficié d'études permettant de connaître les véritables partitions, c'est en ce sens que nous n'utiliserons que les mesures non supervisées sur ces dernières.

Les résultats du PLBS et du MPLBS seront donnés dans un tableau avec un pas faible $\Delta = 0,025$. La ligne des abscisses représente le pourcentage de barrages alors que l'ordonnée représente le score avec la matrice de fréquence relative au nombre de barrages. Pour le PLBS, nous choisirons 3 intervalles, $\Delta_{[0.0;0.33]}$; $\Delta_{[0.33;0.66]}$ et $\Delta_{[0.66;1.0]}$, en utilisant un pas $\Delta = 0.025$.

Dans la suite de cette étude, nous avons utilisé une machine *Predator G3, IntelCore^{TM} i5 processeur 4440, 3,60 Ghz avec 16 GO de RAM*. Le développement des algorithmes est en Python 2.7.

Stabilisation du CDLP

L'une des premières études porte sur le nombre de propagations de labels permettant de stabiliser la propagation de labels, avec le CDLP.

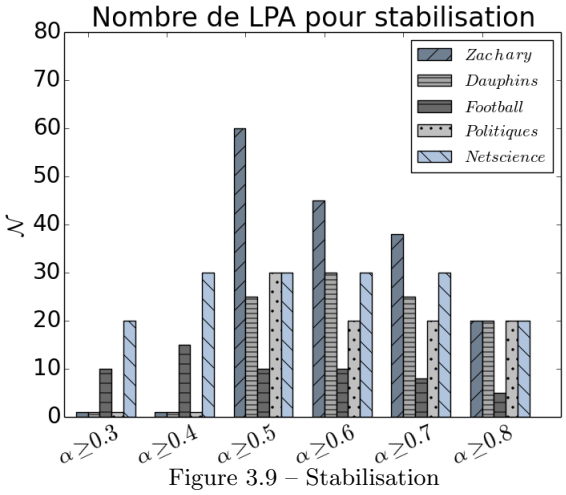


Figure 3.9 – Stabilisation

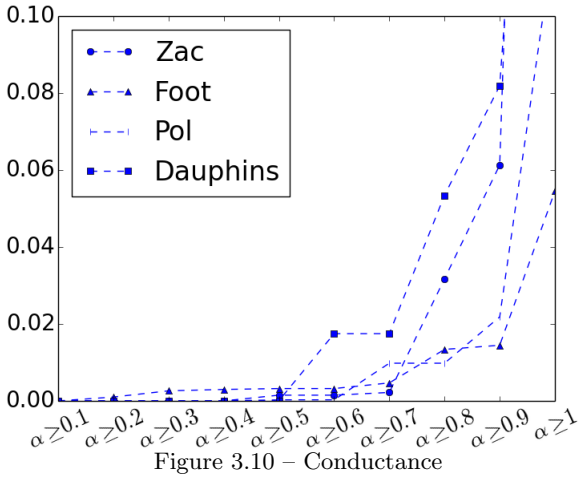


Figure 3.10 – Conductance

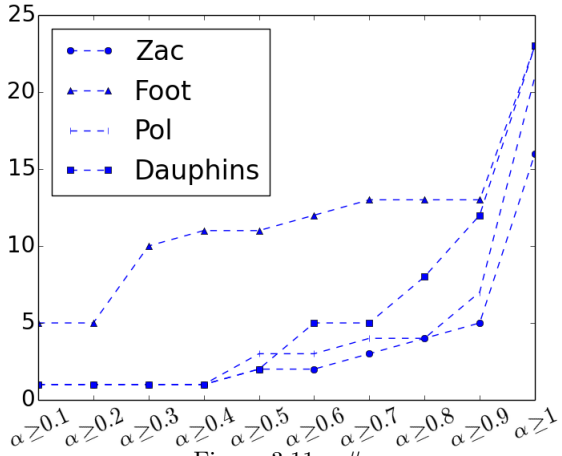


Figure 3.11 – #

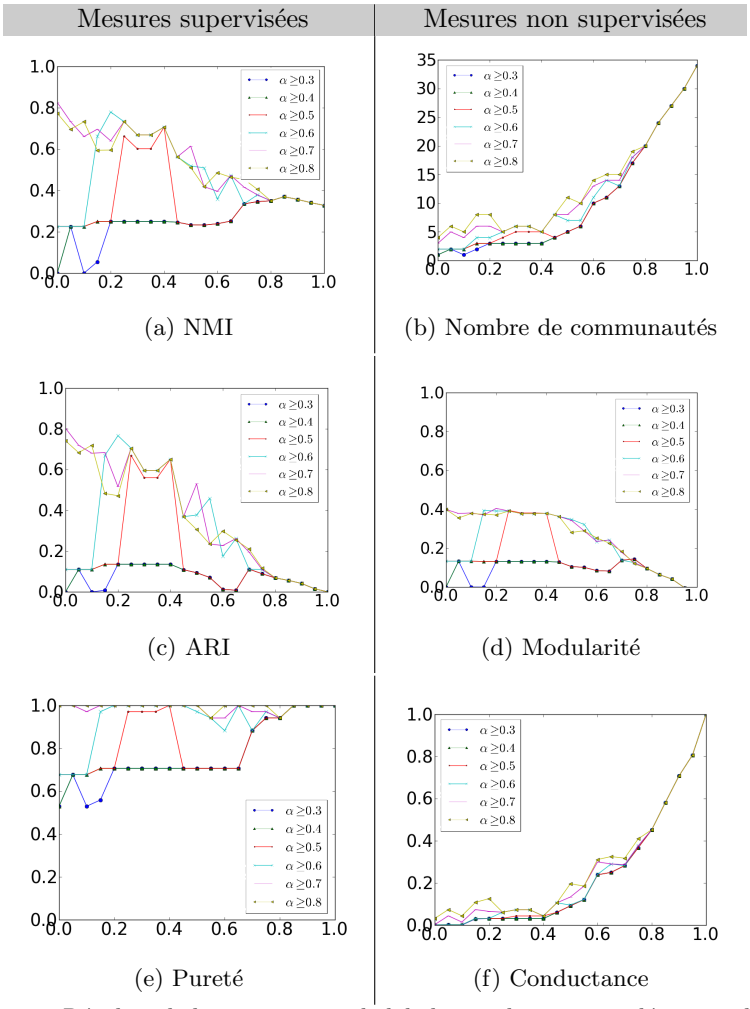
Nous observons que la valeur α joue un rôle sur la stabilisation du LPA, Figure 3.9. Pour $\alpha \geq 0.5$, on voit qu'il faut à peu près 80 itérations pour arriver à stabilisation. Pour $\alpha \geq 0.6$, la stabilisation se fait après 54 propagations de labels. En considérant la conductance pour son information sur la densité intra et inter communauté, Figure 3.10, plus α est important, plus la stabilisation arrivera rapidement, avec plus de communautés de petites tailles, Figure 3.11.

Etude expérimentale sur le MLPBS

Nous avons lancé la propagation de labels avec barrages et détections de cœurs sur des réseaux dont nous connaissions d'avance les vraies communautés, mais aussi sur certains où ce n'est pas le cas. Nous avons choisi pour tous les réseaux un nombre de propagations de labels pour alimenter la matrice de fréquence de $\mathcal{N} = 100$. Le seuil permettant de créer les composantes connexes à partir de la matrice de fréquence se situera dans l'intervalle $\alpha \in [0,3; 0,8]$ avec un pas de $0,1$

Graphe avec vérité de terrains

Pour Zachary, Figure 3.12, nous observons que les résultats sont assez différents selon les valeurs du seuil α. Pour $\alpha \geq \{0.3, ..., 0.5\}$, la mise en place de barrages a une incidence sur la détection de communautés. De base, CDLP ne trouve qu'une grande communauté. C'est entre 20% et 45% de barrages que la méthode obtient de meilleurs résultats. Pour $\alpha \geq 0,6$, on obtient un pic du NMI à 0.78 à 20% de barrages. Pour de très fortes valeurs de α, la présence des barrages ne permet pas d'améliorations notables de la qualité de partitionnement, ce qui est le cas pour $\alpha \geq \{0,7; 0,8\}$. Plus le nombre de barrages augmente, moins la propagation de labels peut s'effectuer. On voit que pour toutes les valeurs de α, un forte augmentation de la conductance a lieu après 40% de barrages, avec une augmentation du nombre de communautés. Après 60% de barrages, on voit une convergence pour toutes les valeurs α, due au très grand nombre de barrages. Pour ce graphe, l'utilisation des barrages semble intéressante pour de faibles valeurs de α.



(a) NMI

(b) Nombre de communautés

(c) ARI

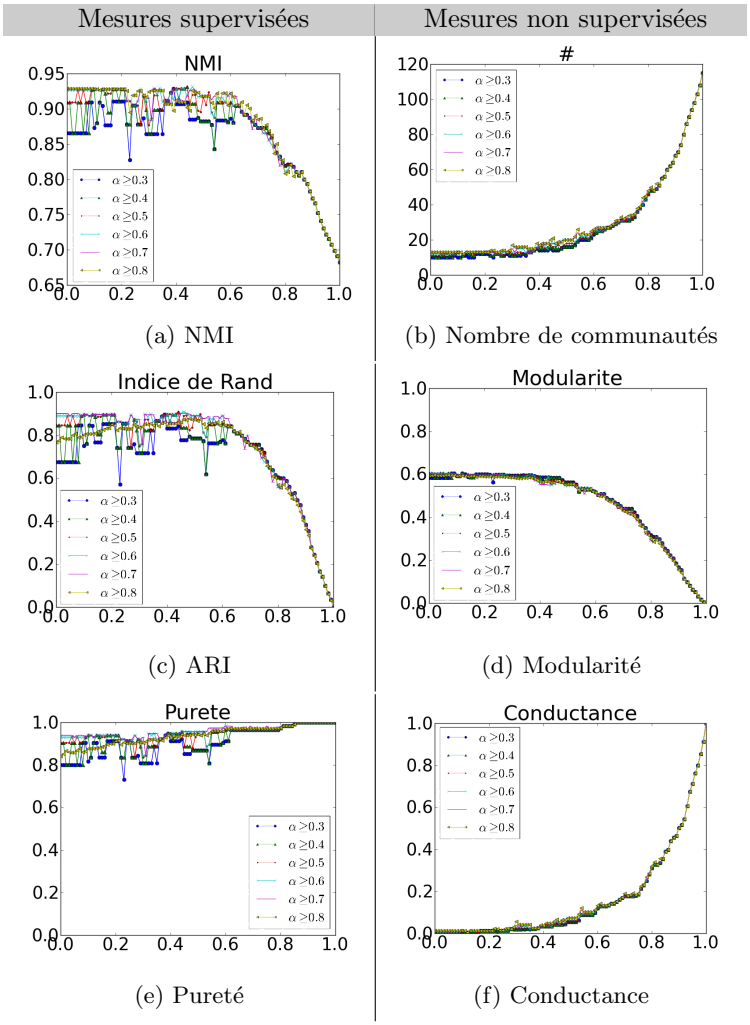
(d) Modularité

(e) Pureté

(f) Conductance

Figure 3.12 – Résultat de la propagation de labels avec barrages et détection de cœurs sur Zachary (l'axe des ordonnées représente les scores des mesures supervisées et non supervisées, l'axe des abscisses représente le pourcentage de barrages).

Pour le club de football, Figures 3.13b, 3.13a, 3.13f, 3.13d 3.13e et 3.13c, la qualité augmente jusqu'à 50% de barrages. La matrice de fréquence stabilise correctement le LPA, sans présence de grands pics. Le nombre de communautés n'augmente que très faiblement jusqu'à 40% et stagne jusqu'à 50%. Les arêtes où les barrages ont été mis sont sur les liens qui lient les communautés entre elles. Après 55% de barrages, la qualité se détériore très rapidement. Selon nos observations, c'est à partir de ce seuil que les barrages se mettent dans des zones fortement denses, c'est-à-dire dans des sous graphes complets, ce qui a pour conséquence de détruire les structures communautaires. La propagation de labels ne peut plus s'effectuer et la qualité de partitionnement se détériore.



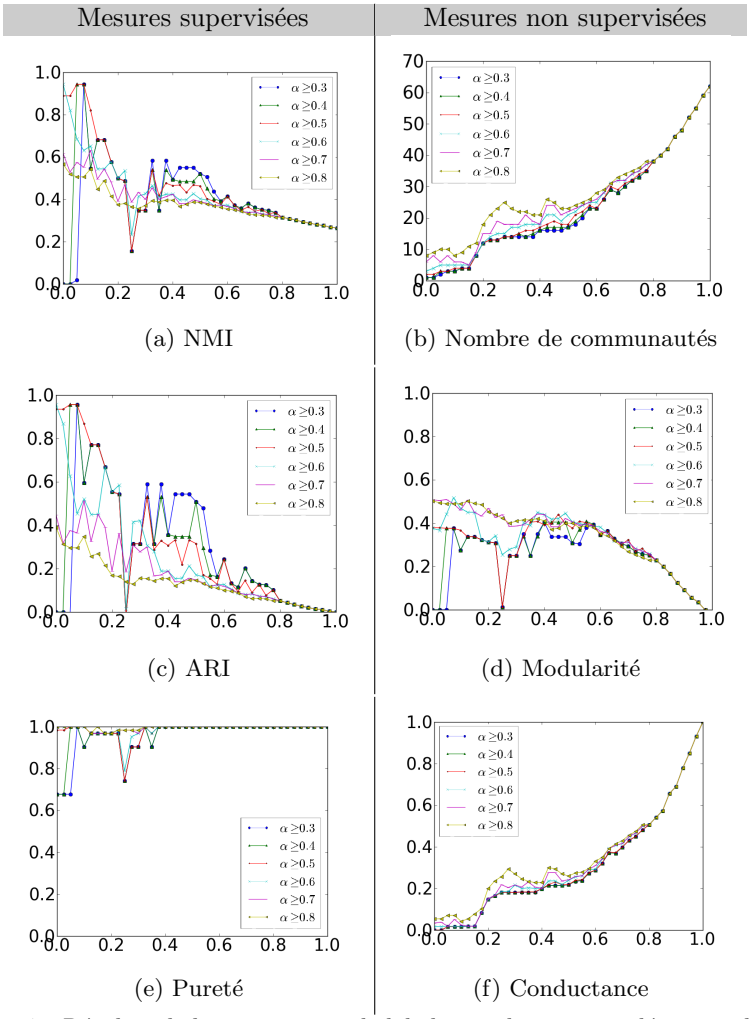
(a) NMI (b) Nombre de communautés

(c) ARI (d) Modularité

(e) Pureté (f) Conductance

Figure 3.13 – Résultat de la propagation de labels avec barrages et détection de cœurs sur le réseau footballistique (l'axe des ordonnées représente les scores de mesures supervisées et non supervisées, l'axe des abscisses représente le pourcentage de barrages).

Pour le réseau de dauphins, Figures 3.14, les meilleurs résultats sont obtenus pour $\alpha \geq \{0.3, 0.4\}$. La qualité se détériore après $\alpha \geq 0.6$. Pour $\alpha \geq 0.3$, jusqu'à 7.5% de barrages, aucune communauté n'est détectée à 10%, deux communautés sont détectées avec un NMI de 0.95. Pour de faibles valeurs de α, la mise en place de barrages peut se révéler bénéfique et permet d'améliorer la qualité de la détection de communautés, ce qui n'est pas le cas pour des valeurs de α très élevées. Pour $\alpha \geq 0.8$, les conductances sont très élevées et l'utilisation de barrages devient inutile.



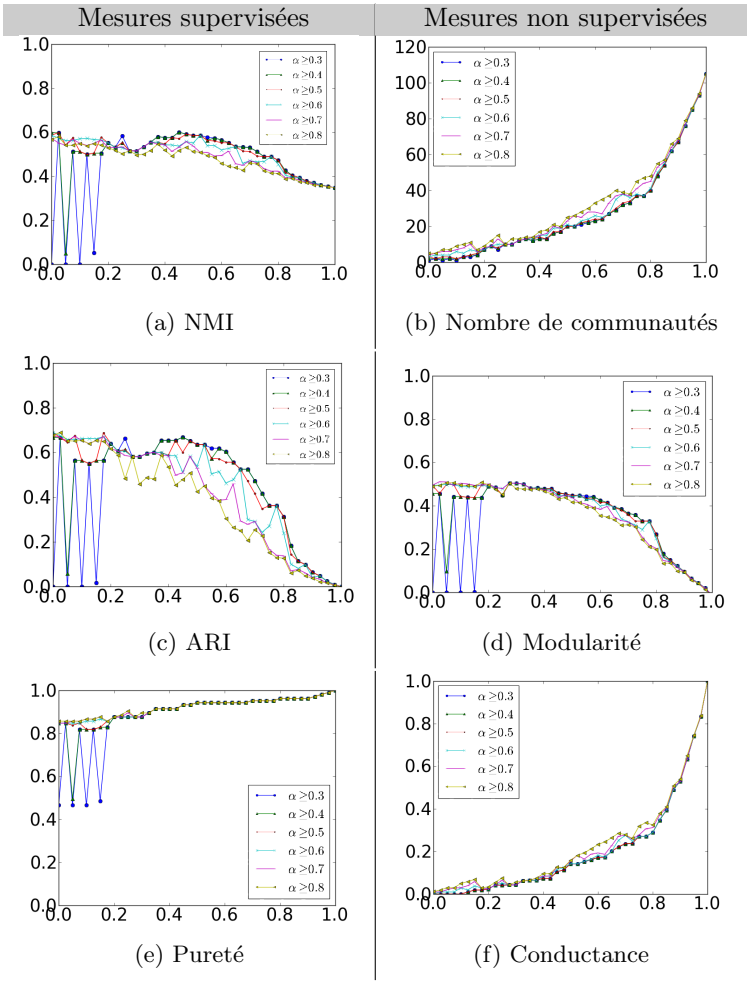
(a) NMI (b) Nombre de communautés

(c) ARI (d) Modularité

(e) Pureté (f) Conductance

Figure 3.14 – Résultat de la propagation de labels avec barrages et détection de cœurs sur le réseau de dauphins (l'axe des ordonnées représente les scores de mesures supervisées et non supervisées, l'axe des abscisses représente le pourcentage de barrages).

Pour les livres politiques de Krebs, Figures 3.15, jusqu'à 10%, le nombre de communautés reste stable avec un bon NMI et un bon ARI. Puis, ce nombre augmente de manière proportionnelle avec le nombre de barrages.



(a) NMI (b) Nombre de communautés

(c) ARI (d) Modularité

(e) Pureté (f) Conductance

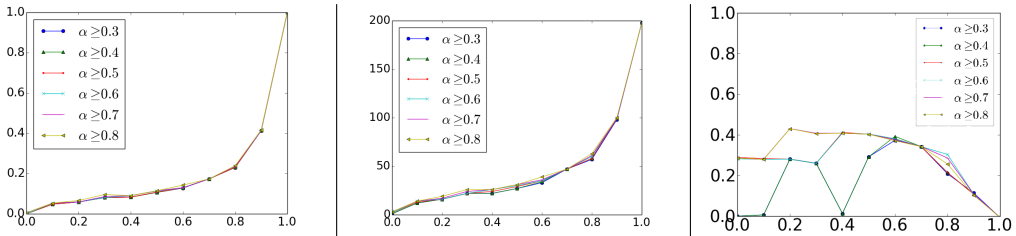
Figure 3.15 – Résultat de la propagation de labels avec barrages et détection de cœurs sur les livres politiques de Krebs (l'axe des ordonnées représente les scores de mesures supervisées et non supervisées, l'axe des abscisses représente le pourcentage de barrages).

Les premiers résultats de ces expérimentations permettent d'affirmer que la qualité de partionnement avec la mise en place de barrages sera fonction de la valeur de α. Si la valeur de α est faible ($\alpha \leq 0,3$), le graphe est considéré comme une seule communauté. L'algorithme se comportera comme si on enlève les arêtes ayant la plus forte centralité d'intermédiarité. Pour une valeur de α comprise entre $0,3$ et $0,7$, notamment pour les graphes sociaux, la mise en place

de barrages permet d'améliorer la qualité de partitionnement jusqu'à un certain seuil. Les observations montrent que le nombre de barrages pour une partition de bonne qualité tourne autour de 25% à 40% sur les réseaux sociaux. Pour un α fort ($\geq 0,8$ pour les graphes sociaux), la mise en place de barrages n'a pas d'effet significatif sur la qualité de partitionnement, les communautés étant de petites tailles. Cependant, un pourcentage trop élevé de barrages détériore la qualité de partitionnement car ces derniers se mettent sur des sous-graphes complets, des cliques.

Graphe sans connaissance des communautés de terrains

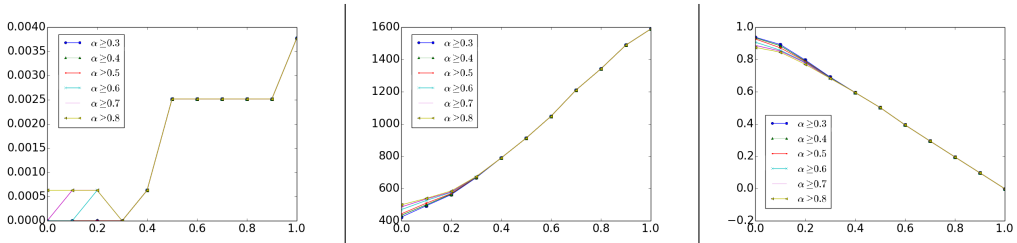
Le réseau de jazz Gleiser et Danon (2003) est un graphe où les nœuds sont les musiciens et les liens le fait qu'ils aient joué ensemble ou pas. Il comporte 198 nœuds pour 5484 arêtes. On observe dans ce cas que α joue un rôle important sur la stabilité et la qualité de l'algorithme. Pour $\alpha \geq \{0,3; 0,4\}$, la modularité est faible avec un pic entre 15% et 20% de barrages puis à 65%. Un faible α ne permet pas de trouver de bonnes structure communautaires. Les meilleurs résultats de la modularité sont atteints avec $\alpha \geq \{0,7; 0,8\}$, où il n'y a pas de présence de cassures. Les communautés étant de plus petites tailles, cela diminue le risque de mauvaise propagation. Les meilleurs résultats sont atteints entre 15% et 20% de barrages. On voit d'ailleurs la conductance plus élevée avec $\alpha \geq \{0,7; 0,8\}$ que pour $\alpha \geq \{0,3; 0,4\}$.



(a) Conductance (b) # (c) Modularité

Figure 3.16 – Résultat de la propagation de labels avec barrages et détection de cœurs sur le réseau de musiciens de jazz avec $\alpha \geq 0,5$ (l'axe des ordonnées représente les scores des mesures supervisées et non supervisées, l'axe des abscisses, le pourcentage de barrages). # dénote le nombre de communautés.

Le réseau Netscience Newman (2006) est un graphe de collaboration dans le domaine des réseaux et leurs applications. Il comprend 1589 nœuds et 2742 arêtes. Les liens se font en utilisant la condition " l'auteur X a écrit de manière conjointe un travail avec l'auteur Y".

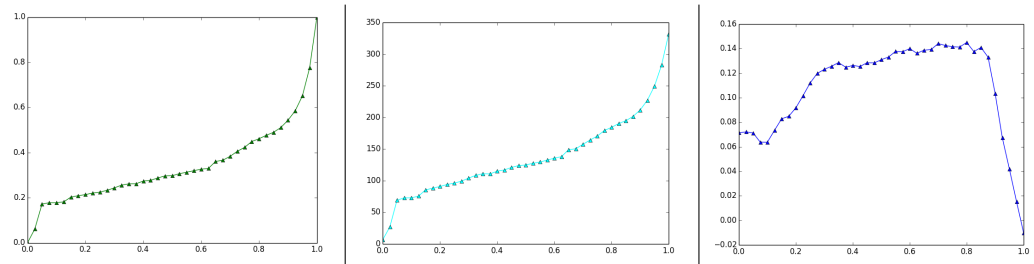


(a) Conductance (b) # (c) Modularité

Figure 3.17 – Résultat de la propagation de labels avec barrages et détection de cœurs sur le réseau de collaboration scientifique Netscience (l'axe des ordonnées représente les scores des mesures supervisées et non supervisées, l'axe des abscisses le pourcentage de barrages). # dénote le nombre de communautés.

Les résultats sont quasi-linéaires en fonction du nombre de barrages. Il s'agit d'un exemple où le fait de mettre des barrages n'apporte pas d'amélioration dans le partitionnement.

Le réseau US-Air 97 est un graphe représentant des infrastructures pour le transport aérien entre le Canada, les Etats-Unis et le Mexique. Il comprend 332 nœuds et 2126 arêtes. Les communautés représentent des infrastructures comme des aéroports, et les arêtes, leurs connexions. Cependant, elles y sont très rares, ce graphe ayant une distribution des degrés des nœuds assez uniforme.



(a) Conductance (b) # (c) Modularité

Figure 3.18 – Résultat de la propagation de labels avec barrages et détection de cœurs sur le réseau de collaboration scientifique Netscience avec $\alpha \geq 0.5$ (l'axe des ordonnées représente les scores des mesures supervisées et non supervisées, l'axe des abscisses, le pourcentage de barrages). # dénote le nombre de communautés.

L'obtention de la modularité maximale requiert près de 80% de barrages. Ceci s'explique par le fait que ce graphe n'a pas de véritables structures communautaires.

De ces analyses, nous pouvons observer que des intervalles sont plus propices à donner de meilleurs résultats que d'autres, ce qui nous a amené à l'élaboration du PLBS.

Etudes expérimentales portant sur le PLBS

MPLBS retournait la meilleure partition sur un intervalle, mais au prix d'une forte complexité. PLBS n'utilise plus qu'une seule matrice de fréquence, dont l'alimentation se fait sur un intervalle spécifique.

Experiences portant sur PLBS						
Algorithmes	Q	Φ	NMI	ARI	Pureté	#
Zac #2						
$\Delta_{[0.0;0.3]}$	0.1314	0.0309	0.2283	0.0908	0.6765	3
$\Delta_{[0.3;0.6]}$	0.378	0.0728	0.5653	0.498	0.9706	6
$\Delta_{[0.6;1.0]}$	0.0736	0.5502	0.3773	0.068	1.0	23
Foot #11						
$\Delta_{[0.0;0.3]}$	0.599	0.012	0.9108	0.8523	0.913	12
$\Delta_{[0.3;0.6]}$	0.5844	0.0419	0.9311	0.9066	0.9565	16
$\Delta_{[0.6;1.0]}$	0.3133	0.3271	0.8309	0.622	0.9913	48
Dauphins #2						
$\Delta_{[0.0;0.3]}$	0.2749	0.0684	0.5466	0.5968	0.9032	3
$\Delta_{[0.3;0.6]}$	0.3924	0.4098	0.4558	0.2991	1.0	18
$\Delta_{[0.6;1.0]}$	0.281	0.6943	0.3283	0.0708	1.0	35
Pol #3						
$\Delta_{[0.0;0.3]}$	0.4961	0.0288	0.5649	0.6713	0.8571	6
$\Delta_{[0.3;0.6]}$	0.4594	0.104	0.6006	0.6684	0.9333	16
$\Delta_{[0.6;1.0]}$	0.2122	0.3129	0.4344	0.1941	0.9619	45

Tableau 3.1 – Résultats du PLBS sur des réseaux connus de la littérature

A partir du tableau 3.1, les meilleurs résultats pour les réseaux Zac,Pol et Foot sont en prenant l'intervalle $[0.3, 0.6]$ (en faisant donc varier le nombre de barrages de 30 à 60 %.), permettant l'alimentation de la matrice de fréquence. Pour les dauphins, il s'agit de l'intervalle $[0.0, 0.3]$. Nous notons que plus les bornes de l'intervalle ont des valeurs importantes (avec un pourcentage de barrages assez fort), plus le nombre de communautés est important. Cela s'explique du fait que la propagation ne peut plus s'effectuer dans certaines régions du graphe. Par voie de conséquence, plus le nombre de communautés est important, plus les densités de ces dernières augmentent, ce qui entraîne une augmentation de la conductance. On observe que la qualité des communautés se dégrade, avec un NMI et un ARI tendant vers 0.

Etudes comparative entre le PLBS et CDLP

Nous proposons d'analyser le comportement du PLBS vis-à-vis de CDLP sur de petits graphes réels. L'objectif est d'observer l'impact du pourcentage de barrages lors de l'alimentation de la matrice du PLBS sur la qualité de partitionnement, d'observer la sensibilité des méthodes vis-à-vis du seuil α (qui permet de créer le graphe seuillé à partir de la matrice de fréquence) et de voir si augmenter le nombre de propagations de labels a un impact sur les partitions résultantes.

Nous proposons d'utiliser le PLBS alimenté par 10%, 20% puis par 30% de barrages avec comme fonction supervisée le NMI, et avec $\mathcal{N} = 100$, $\mathcal{N} = 500$ puis $\mathcal{N} = 1000$. Les Figures 3.19,3.20,3.21 et 3.22 montrent les résultats respectivement sur Zachary, les dauphins, le réseau footballistique et le réseau de livres politiques vis-à-vis du seuil α qui permet de créer le graphe seuillé pour la détection des communautés.

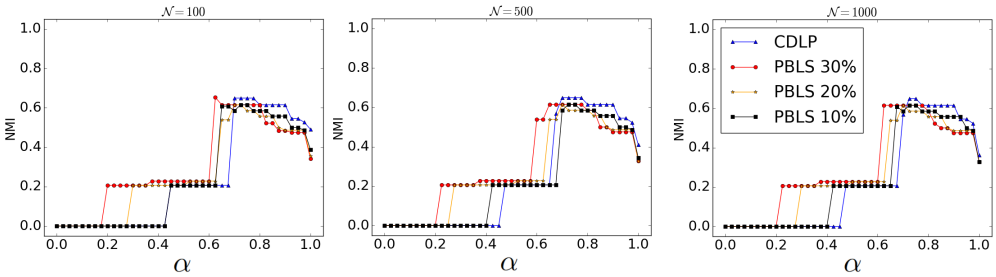


Figure 3.19 – Etude comparative entre PLBS et CDLP sur Zachary

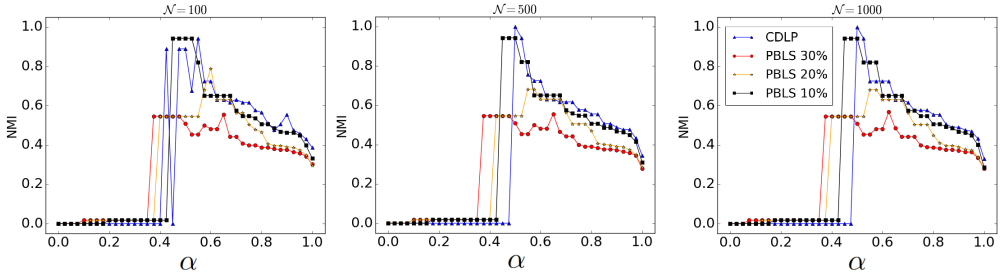


Figure 3.20 – Etude comparative entre PLBS et CDLP sur les dauphins

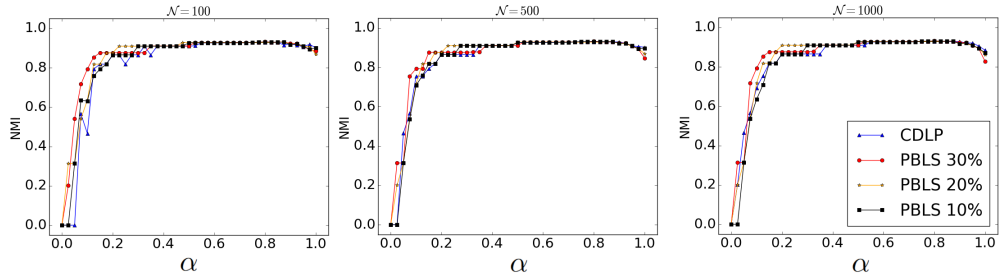


Figure 3.21 – Etude comparative entre PLBS et CDLP sur le réseau footballistique

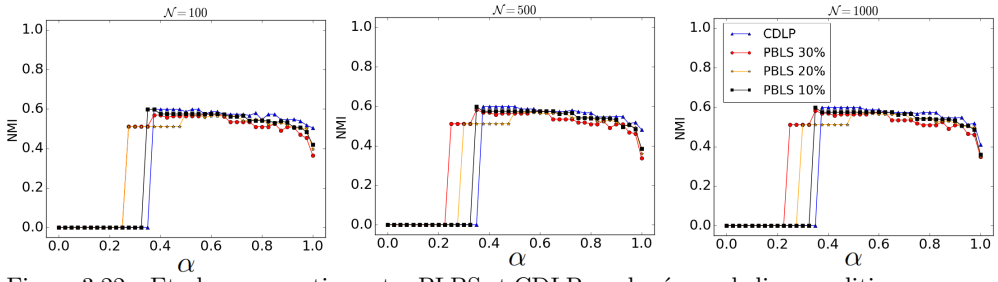


Figure 3.22 – Etude comparative entre PLBS et CDLP sur le réseau de livres politiques

D'après les expérimentations, PLBS est plus sensible aux valeurs de α que ne l'est CDLP. En effet, les premières communautés détectées pour le PLBS sont à partir de $\alpha \geq 0.2$ pour Zachary avec 30% de barrages alors que les premières communautés avec CDLP ne sont détectées qu'à partir de $\alpha \geq 0.4$. Pour les livres politiques, c'est à partir de $\alpha \geq 0.20$ que les premières communautés sont détectées avec PLBS et $\alpha \geq 0.38$ pour CDLP. De même, pour le réseau de dauphins, avec $\alpha \geq 0.22$ PLBS détecte ses premières communautés alors que CDLP détecte les siennes à partir de $\alpha \geq 0.40$. Les communautés sont détectées plus vite avec un fort pourcentage de barrage (30%).

Augmenter $\mathcal{N}$ ne stabilise pas davantage le PBLS. Pour le réseau de dauphins, augmenter $\mathcal{N}$ stabilise plus le CDLP où des fluctuations avec $\mathcal{N} = 100$ apparaissent à $\alpha = 0.45$ et$\alpha \geq 0.58$, que l'on ne voit plus pour $\mathcal{N} = 500$ et $\mathcal{N} = 1000$. CDLP montre des résultats parfois meilleurs que PBLS notamment pour Zachary, mais sur des intervalles très petits. PBLS donne en moyenne de meilleurs résultats que CDLP. Alimenter les matrices de 10% à 30% de barrages donne des résultats très similaires pour PLBS pour ces types de réseaux.

Visualisation des matrices PLBS et CDLP

Dans cette section, nous nous focalisons sur l'alimentation des matrices de fréquence, mais également sur l'ajout de barrages pour l'alimentation de la matrice du PLBS.

Pour le CDLP, nous lançons séquentiellement $\mathcal{N}$ propagations de labels, indépendantes les unes des autres. A chaque propagation de label, les éléments de la matrice de fréquence sont incrémentés de 1. Lorsque toutes les propagations de labels ont été effectuées, il y a normalisation en divisant chaque élément par $\mathcal{N}$. Pour PLBS, nous verrons que la mise en place de barrages va modifier significativement la matrice de fréquence. L'idée est également d'observer si des zones, avec un pourcentage fort de barrages, apparaissent. Nous exposons le CDLP et le PLBS avec différents pourcentages de barrages sur le graphe de Karaté Figure 3.23, sur celui des dauphins Figure 3.24, sur le réseau footballistique 3.25, sur le réseau de livres politiques 3.26 et sur le réseau de jazz 3.27.

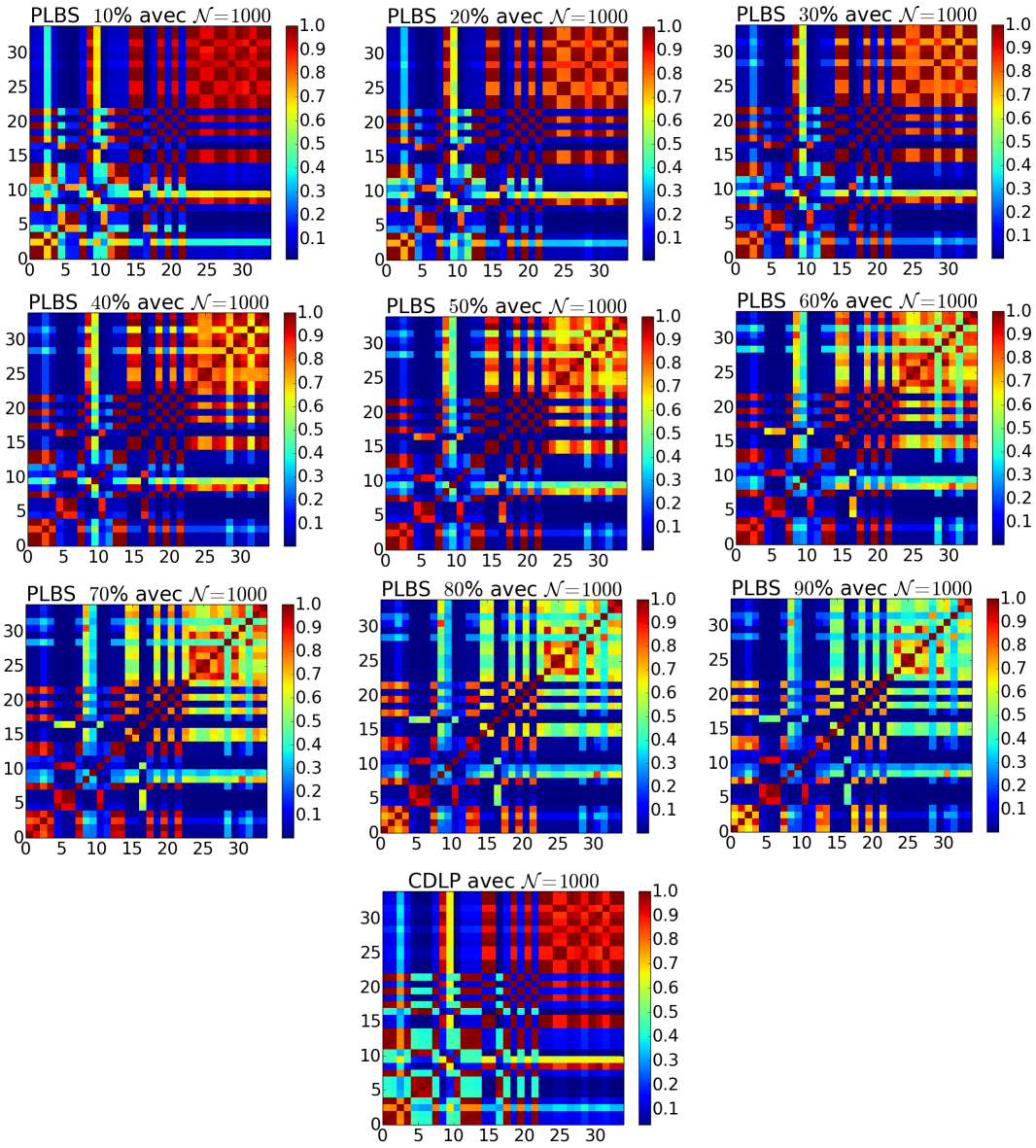


Figure 3.23 – Matrices du PLBS et du CDLP sur Zachary

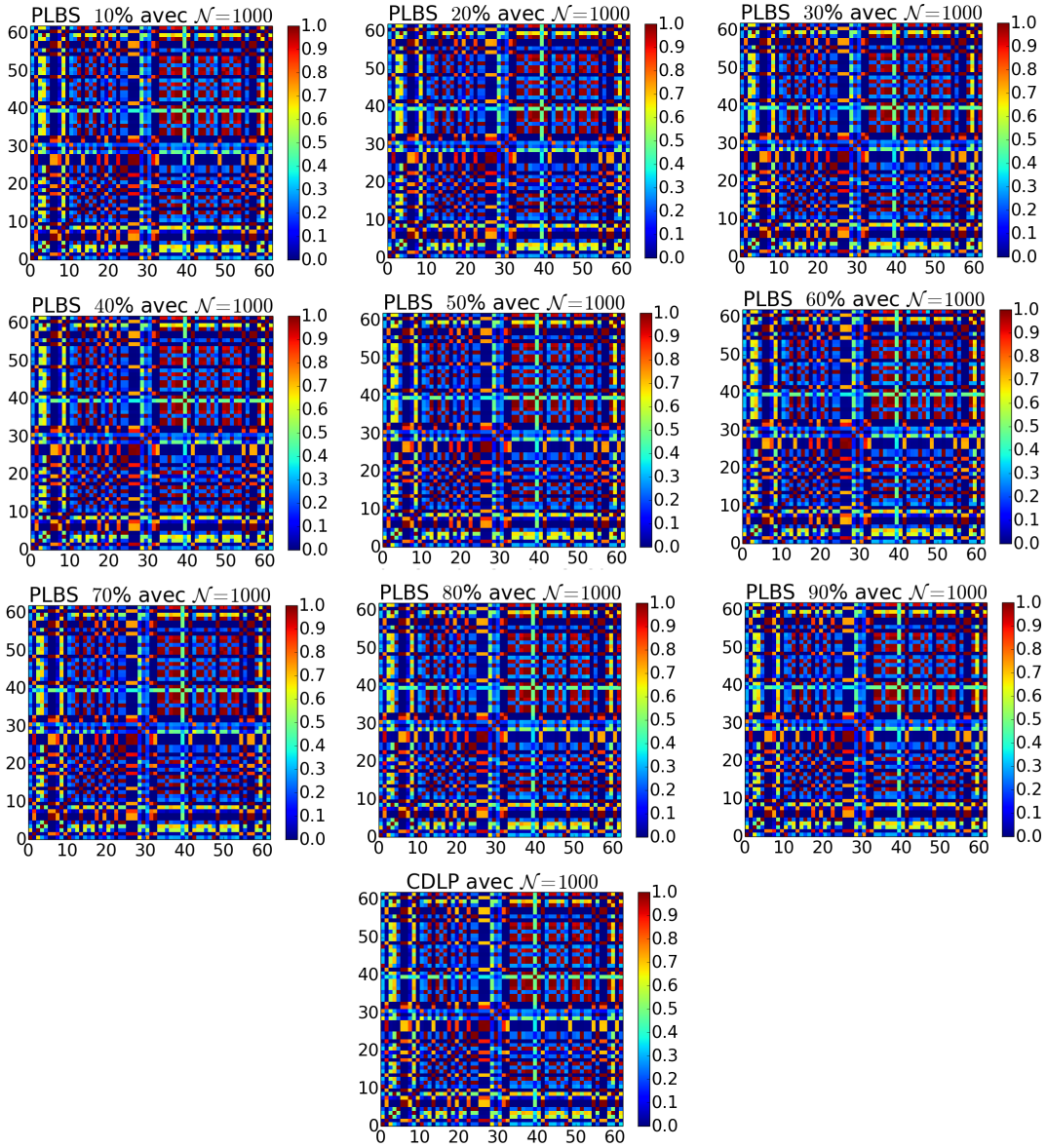


Figure 3.24 – Matrices du PLBS et du CDLP sur le réseau de dauphins

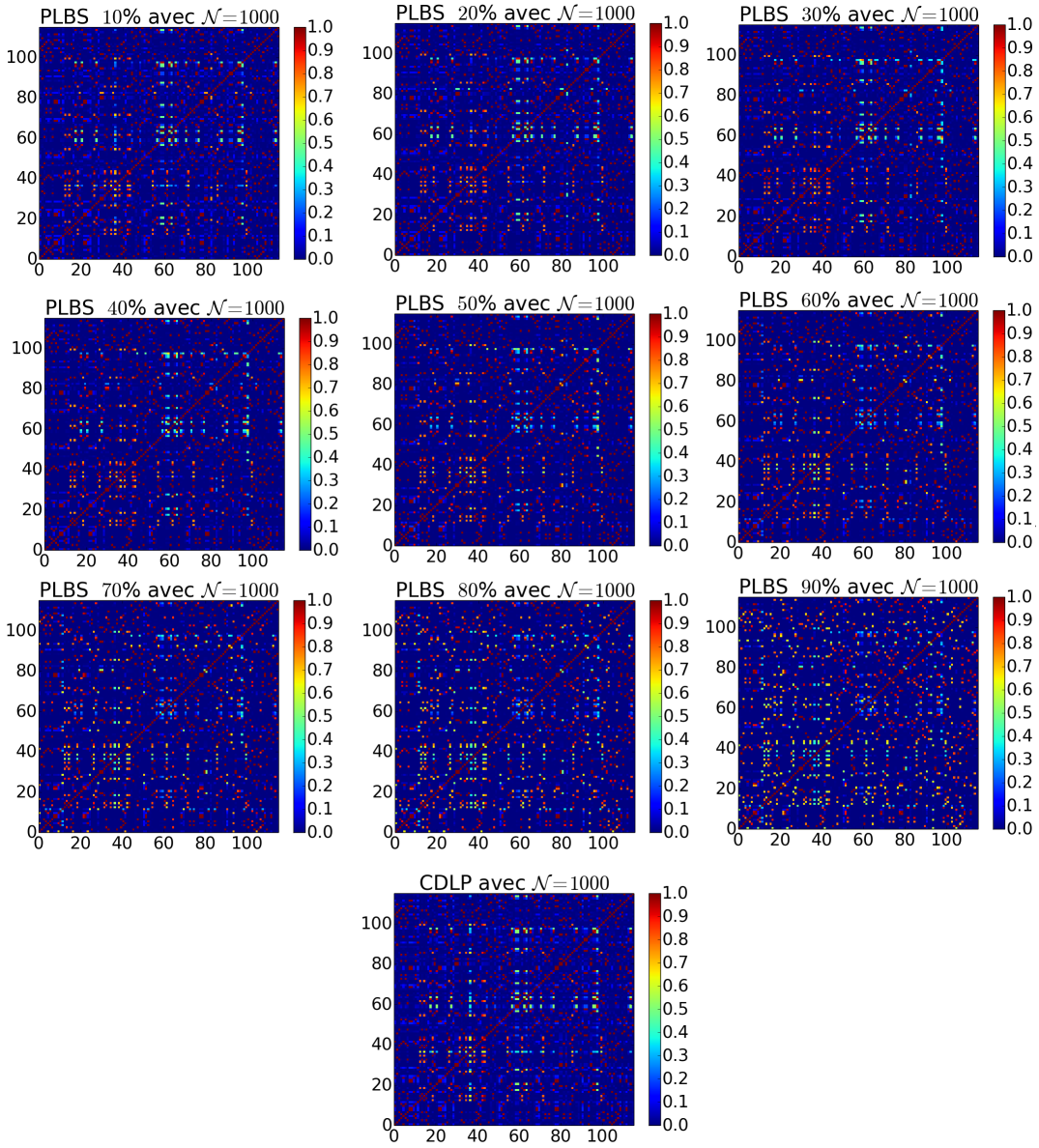


Figure 3.25 – Matrices du PLBS et du CDLP sur le réseau footballistique

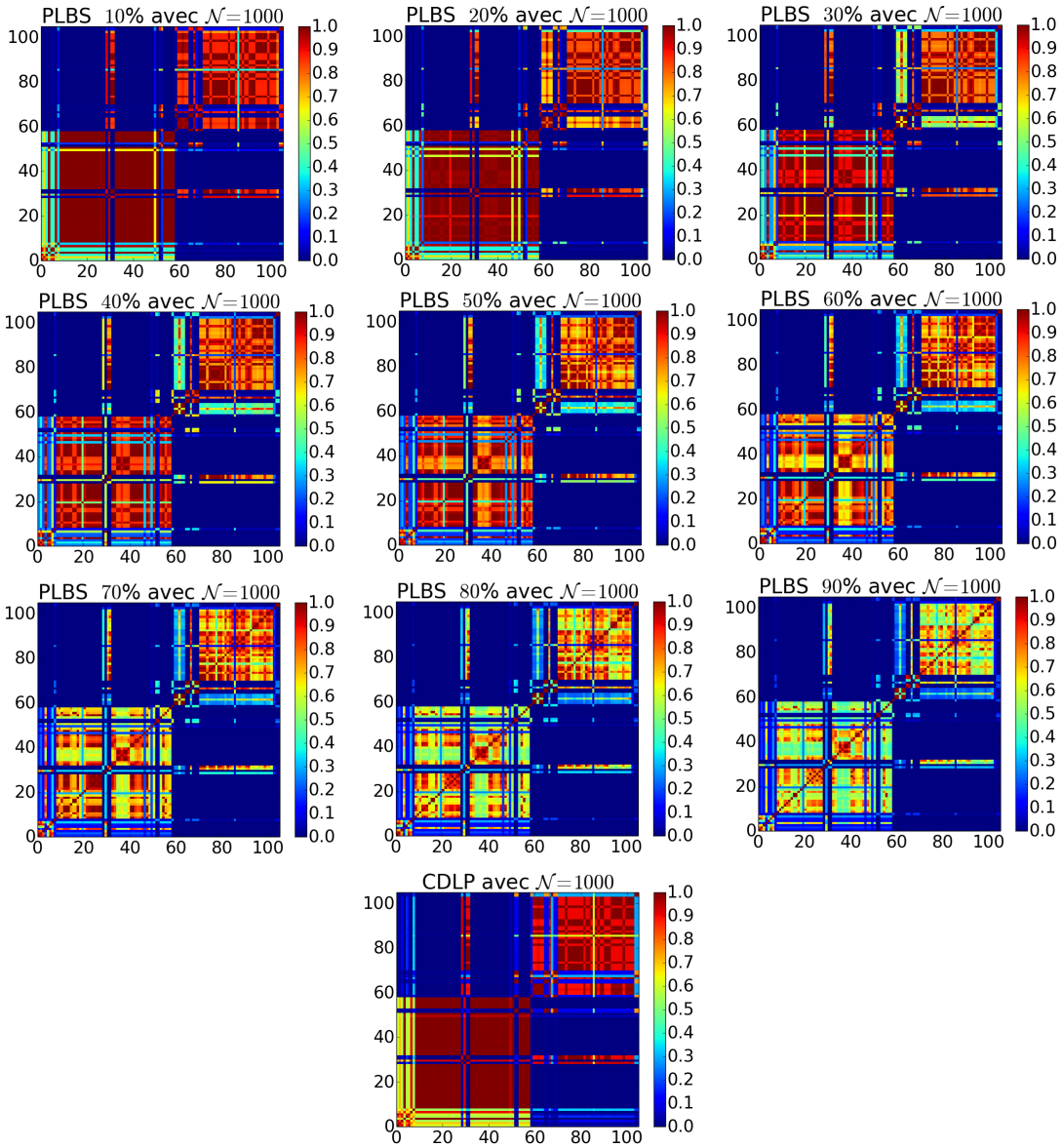


Figure 3.26 – Matrices du PLBS et du CDLP sur les livres politiques

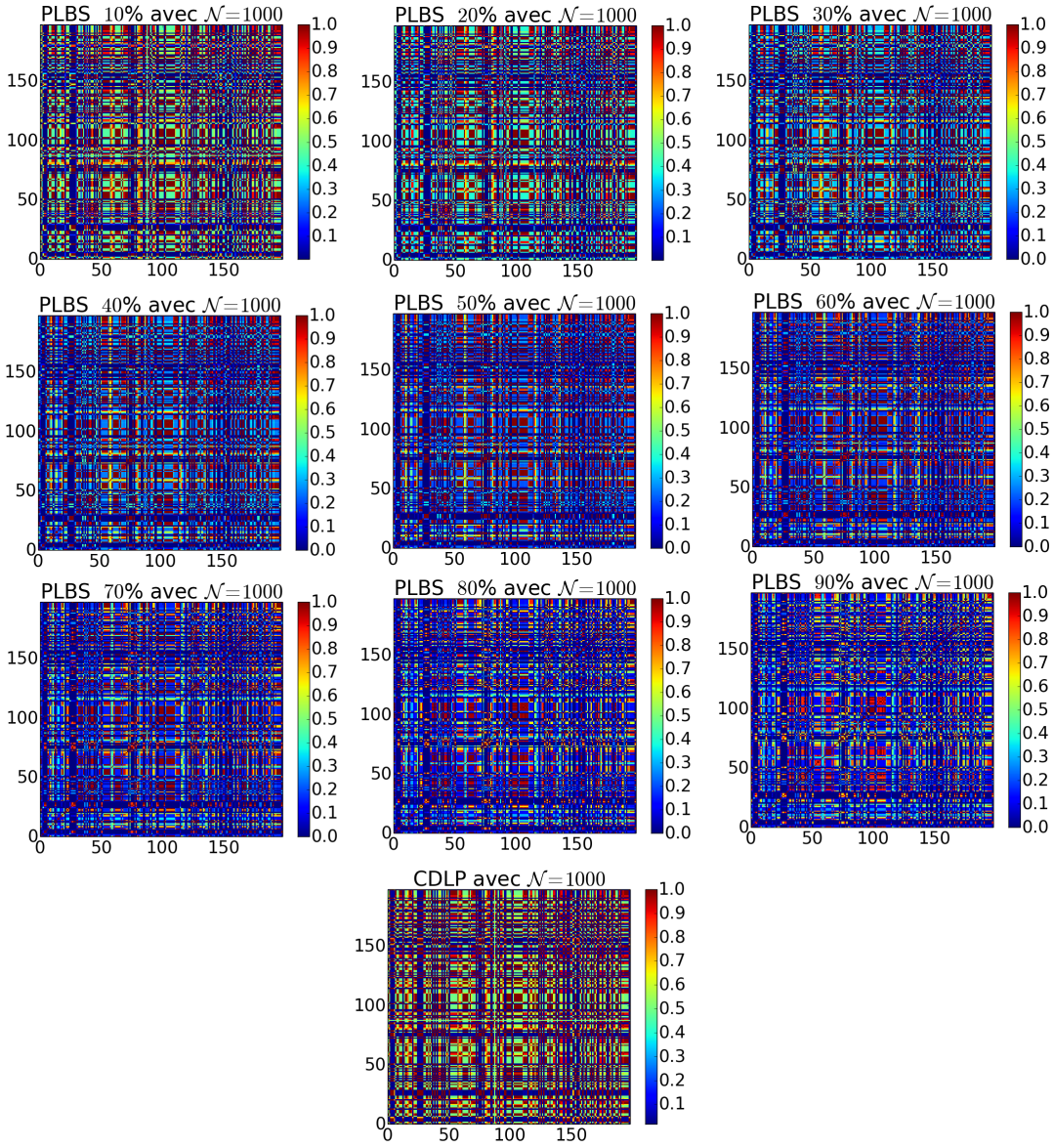


Figure 3.27 – Matrices du PLBS et du CDLP sur le réseau de jazz

Les matrices résultantes sont des matrices par bloc comme nous pouvons le voir pour le club de Karaté, les dauphins et les livres politiques. En observant les matrices de CDLP, nous pouvons voir des groupes de nœuds qui ont une forte probablité d'être ensemble. Par exemple, en observant la matrice du club de Karaté, nous voyons de petites matrice rouges, qui pour le PBLS, en augmentant les barrages, restent présentes. Ces liens qui ont une forte valeur sont significatifs de la présence de cœurs. C'est par exemple le cas pour le réseau footballistique

où les points rouges, sont les paires de nœuds se trouvant très fréquemment dans les mêmes communautés.

En augmentant le pourcentage de barrages, les matrices deviennent presque diagonales, ce qui est le cas sur tous nos exemples.

Etude sur le temps d'exécution

L'algorithme est composé de plusieurs parties dont les temps d'exécution sont différents les uns des autres. Nous rappelons que nous avons utilisé une machine *Predator G3, IntelCore^{TM} i5 processeur 4440, 3,60 Ghz avec 16 GO de RAM*. Le développement des algorithmes est en Python 2.7.

Le temps d'exécution du PLBS nécessite 3 étapes :
— le temps de calcul de la centralité d'intermédiarité (Δ_{CI})
— le temps pour mettre les barrages, l'allocation d'une matrice de fréquence et son alimentation par les différentes propagations de labels ($\Delta_{matrice_{LPA}}$)
— le temps pour la détection de composantes connexes correspondant à nos communautés (Δ_{CC})
Cela peut se schématiser par la formule suivante :

$$\Delta_{T_{PLBS}} = \Delta_{CI} + \Delta_{matrice_{LPA}} + \Delta_{CC} \tag{3.1}$$

Les parties de lecture et d'écriture n'ont pas été mises dans la mesure où elles ne considèrent pas le corps de l'algorithme. Nous obtenons les résultats suivants :

Pour l'algorithme PLBS, nous prenons $\mathcal{N} = 100$, un intervalle de $\Delta_{[x,y]} = \Delta_{[0.0,0.3]}$ avec un pas $\Delta = 0.025$.

Temps d'exécution en secondes					
Réseau	$\|V\|$ et $\|E\|$	Δ_{CI}	$\Delta_{Matrice(LPA)}$	Δ_{CC}	$\Delta_{T_{PLBS}}$
Zac	$34 \setminus 78$	0.0004	0.280	0.003	0.284
Dauphins	$62 \setminus 159$	0.002	0.587	0.019	0.608
Pol	$105 \setminus 441$	0.008	4.158	0.217	4.382
Foot	$115 \setminus 615$	0.010	4.309	0.276	4.6
Jazz	$198 \setminus 5484$	0.125	15.432	0.077	15.634
US-Air 97	$332 \setminus 2126$	0.095	16.935	0.226	17.256
Netscience	$1589 \setminus 2742$	0.061	91.72	1.218	93.098

Tableau 3.2 – Résultats sur le temps d'éxecution du PLBS

Nous observons que la partie prenant le plus de temps concerne l'alimentation de la matrice par les $\mathcal{N}$ propagations de labels. Pour Netscience, le temps commence à devenir conséquent avec près de 93.1 secondes. Le calcul de la centralité d'intermédiarité est assez rapide sur Igraph, bien que sa complexité soit élevée.

MPLBS utilise quant à elle K matrices différentes avec différents niveaux de barrages. Le temps d'exécution du MPLBS comprend 4 étapes :

— le temps de calcul de la centralité d'intermédiarité (Δ_{CI})
— le temps pour mettre les barrages, l'allocation des K matrices de fréquence et leur alimentation par les différentes propagation de labels ($\Delta_{matrice_{LPA}}$)
— le temps pour la détection de composantes connexes correspondant à nos communautés (Δ_{CC})
— le temps de calcul de la fonction de qualité choisie, la modularité Δ_Q ou la conductance Δ_Φ

Pour l'algorithme MPLBS, nous prenons un intervalle $\Delta = 0.025$ et $\mathcal{N} = 100$. Le temps d'exécution peut se schématiser par la formule suivante :

$$\Delta_{T_{MPLBS}} = \Delta_{CI} + \Delta_{matrice_{LPA}} + \Delta_{CC} \tag{3.2}$$

Pour le calcul du temps global, nous sommerons le temps de calcul de la modularité et celui de la conductance.

Temps d'exécution en secondes							
Réseau	$\|V\|$ et $\|E\|$	Δ_{CI}	$\Delta_{Matrice(LPA)}$	Δ_{CC}	Δ_Q	Δ_Φ	$\Delta_{T_{MPLBS}}$
Zac	$34 \setminus 78$	0.0005	0.642	0.0006	0.0005	0.068	0.712
Dauphins	$62 \setminus 159$	0.084	1.711	0.001	0.0005	0.002	1.8
Pol	$105 \setminus 441$	0.018	4.355	0.002	0.0007	0.002	4.378
Foot	$115 \setminus 615$	0.022	3.614	0.002	0.0009	0.958	4.597
Jazz	$198 \setminus 5484$	0.147	21.676	0.006	0.0011	0.639	22.47
US-Air 97	$332 \setminus 2126$	0.095	98.381	0.042	2.631	0.484	101.632
Netscience	$1589 \setminus 2742$	0.061	376.167	0.02	0.007	73.981	450.236

Tableau 3.3 – Résultats sur le temps d'exécution du MPLBS

En observant les Tableaux 3.2 et 3.3, nous voyons que le temps d'exécution du MPLBS est beaucoup plus important que celui du PLBS. Cela vient du fait qu'il y a $\frac{1}{\Delta}$ matrices de fréquence à alimenter, d'après les notations de l'Algorithme 3. C'est également cette alimentation qui prend le plus de temps par rapport aux autres traitements, qui prennent beaucoup moins de temps. Le temps de calcul de la conductance et de la modularité ainsi que le temps de calcul des composantes connexes sont très faibles. La taille de la matrice à alimenter est fonction de la taille du graphe considéré. Plus le graphe aura de nœuds, plus le temps d'alimentation sera important. Une étude comparative avec les algorithmes de détection de communautés issus de la littérature pourra être trouvée à la sous-section 3.3.

Etude de la corrélation entre la centralité d'intérmédiarité et la matrice de fréquence du CDLP

Les études précédentes ont montré que l'ajout de barrage en utilisant l'information de la centralité d'intermédiarité pouvait dans certains cas améliorer la qualité de partitionnement. Cependant, la question est de savoir si mettre des barrages sur les liens de forte centralité d'intermédiarité ne serait pas équivalent à mettre des barrages en fonction des valeurs des éléments de la matrice de

fréquence, c'està-dire des éléments ayant une faible probabilité d'être ensemble. Pour répondre à cette question, nous étudions expérimentalement la corrélation entre les valeurs de la matrice de fréquence p_{ij} et la centralité d'intermédiarité fondée sur les arêtes. Nous faisons différents tests avec un nombre de propagations de labels différent pour voir si cela peut avoir une influence sur le résultat, notamment avec $\mathcal{N} = 100$, $\mathcal{N} = 500$ et $\mathcal{N} = 1000$.

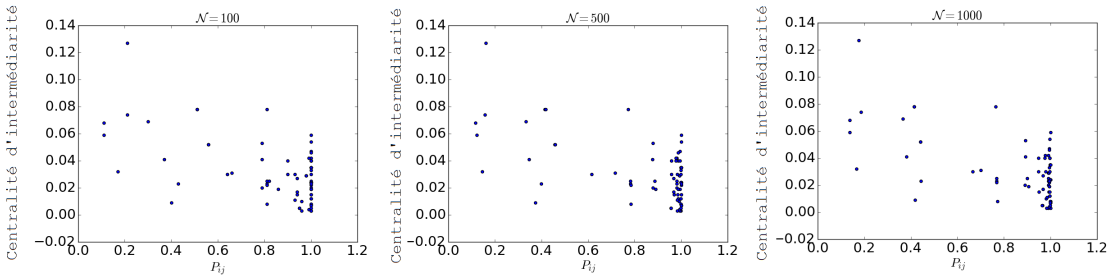


Figure 3.28 – Etude de la corrélation entre la centralité d'intermédiarité et les éléments de la matrice de fréquence sur Zachary

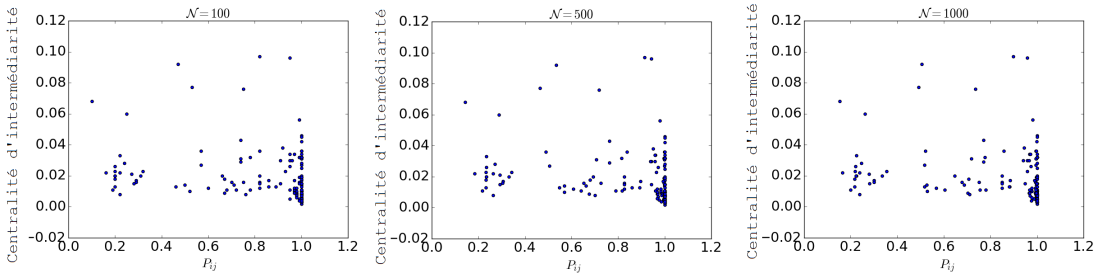


Figure 3.29 – Etude de la corrélation entre la centralité d'intermédiarité et les éléments de la matrice de fréquence sur le réseau de dauphins

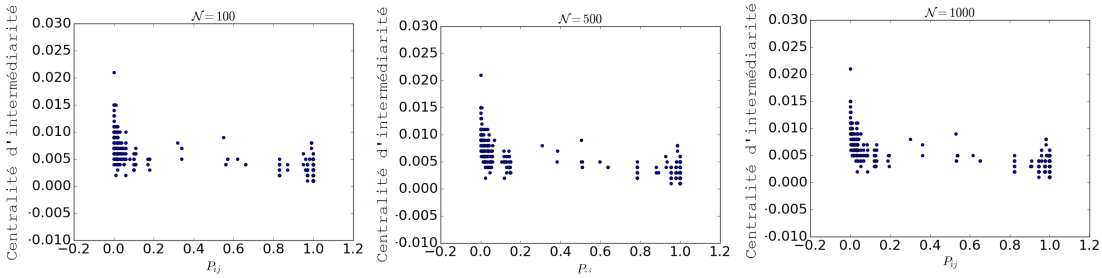


Figure 3.30 – Etude de la corrélation entre la centralité d'intermédiarité et les éléments de la matrice de fréquence sur le réseau footballistique

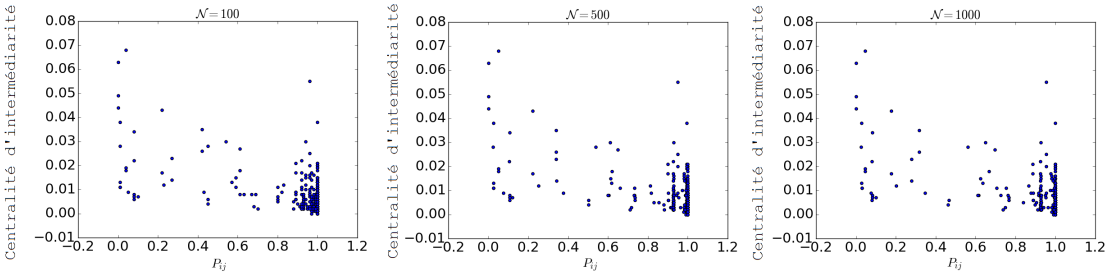


Figure 3.31 – Etude de la corrélation entre la centralité d'intermédiarité et les éléments de la matrice de fréquence sur le réseau de livres politique

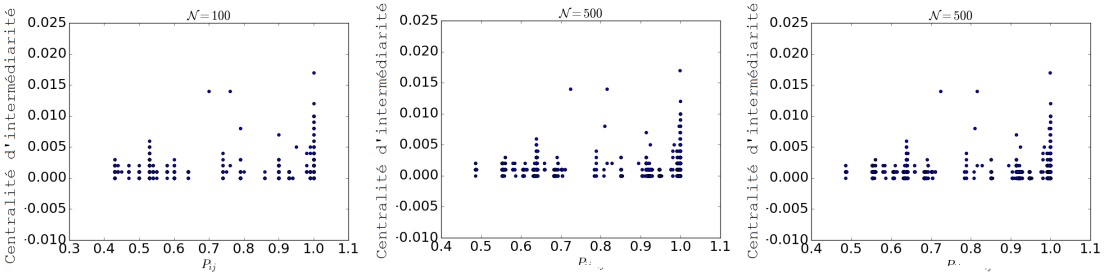


Figure 3.32 – Etude de la corrélation entre la centralité d'intermédiarité et les éléments de la matrice de fréquence sur le réseau de jazz

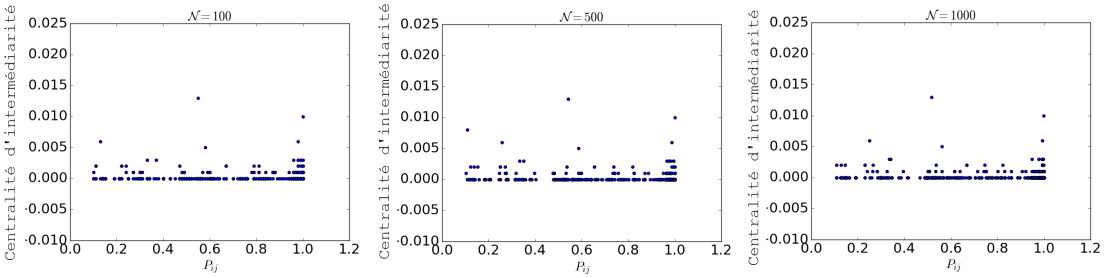


Figure 3.33 – Etude de la corrélation entre la centralité d'intermédiarité et les éléments de la matrice de fréquence sur le réseau de collaboration scientifique

Réseaux	$\mathcal{N}=100$	$\mathcal{N}=500$	$\mathcal{N}=1000$
Zachary	-0.573	-0.592	-0.592
Dauphins	-0.226	-0.229	-0.216
Foot	-0.756	-0.762	-0.761
Livres politiques	-0.570	-0.575	-0.571
Jazz	-0.186	-0.189	-0.189
Netscience	-0.170	-0.197	-0.176

Tableau 3.4 – Coefficient de Pearson entre la centralité d'intermédiarité et les éléments de la matrice de fréquence.

Les figures ci-dessus et le Tableau 3.4 montrent le degré de corrélation entre la centralité d'intermédiarité et les éléments de la matrice de fréquence du CDLP, soit les p_{ij}. Nous aurions pu nous attendre à ce que les liens ayant une forte centralité d'intermédiarité soient des arêtes avec un faible p_{ij}, c'est-à-dire dont les sommets auraient une faible probabilité d'être dans une même communauté et inversement. Cependant, les coefficients de Pearson montrent qu'il y a une faible anti-corrélation (parfois même qu'il n'y a pas de corrélation du tout comme pour Netscience avec -0.170) entre les deux séries de mesures. De plus le Tableau 3.4 montre que le fait d'augmenter le nombre de propagations de labels pour stabiliser la méthode n'améliore pas la corrélation entre les deux séries de mesures.

3.1.4 Conclusion sur les algorithmes de propagation de labels avec barrages et détection de cœurs

Nous avons exposé de nouveaux algorithmes fondés sur la propagation de labels. Nos expérimentations ont montré que notre méthode hybride, liant propagation de labels et barrages issus de l'intermédiarité des arêtes permettait l'obtention de résultats satisfaisants.

Le choix du seuil α dépend de la topologique du graphe considéré. Pour nos exemples, et ce qui concerne le CDLP, un α faible ne permet pas de détecter de communautés. Ainsi c'est pour Zachary, c'est avec $\alpha \geq 0.5$ que les premières communautés sont détectées, il en est de même pour le réseau de dauphins et le réseau de livres politiques. Pour le réseau footballistique et le réseau de collaboration scientifique, les premières communautés sont détectées avec $\alpha \geq 0.3$. Plus le seuil α augmente, plus le nombre de LPA nécessaires est faible. Sur le club de karaté, il faut à peu près une soixantaine de LPA pour rendre la méthode déterministe et seulement une vingtaine avec $\alpha \geq 0.8$. Ce constat peut être réalisé pour tous les réseaux que nous avons étudiés.

Concernant le PLBS, l'ajout de barrages stabilise davantage l'algorithme que le CDLP. En mettant des barrages sur les liens, certaines propagations de labels ne peuvent plus s'effectuer. En augmentant le nombre de barrages à l'alimentation du PLBS, la matrice de fréquence se rapproche d'une matrice diagonale. Nous avons observé qu'avec un pourcentage de barrages assez fort, (60%,70%,80% et 90%) des paires de nœuds apparaissent, ce qui montre la présence de cœurs. Ce constat peut être réalisé pour tous les réseaux que nous avons étudiés.

L'ajout de barrages permet de détecter plus rapidement les communautés, c'est-à-dire avec un seuil α plus faible, par exemple, pour le club de karaté, En effet, les premières communautés détectées pour PLBS le sont à partir de $\alpha \geq 0.2$ pour Zachary avec 30% de barrages alors que les premières communautés avec CDLP ne sont détectées qu'à partir de $\alpha \geq 0.5$. Sur l'ensemble du spectre du seuil α PLBS donne de meilleurs résultats que CDLP en termes de

qualité de partitionnement.

Nous avons mené une étude expérimentale qui a montré qu'il n'y avait pas de fortes corrélations entre la centralité d'intermédiarité des arêtes et les valeurs de la matrice de fréquence du CDLP. Par exemple, le coefficient de Pearson entre les séries de mesures pour le réseau de dauphins n'est que de -0.216.

Les algorithmes MPLBS, PLBS et CDLP sont déterministes par rapport à des algorithmes tels que Louvain ou le LPA. Cependant, un critère de partitionnement est nécessaire pour couper le dendrogramme, c'est en ce sens que des mesures non supervisées comme la modularité sont utilisées. En utilisant comme critère de partionnement la modularité, nous verrons dans la section portant sur l'analyse comparative 3.3, que nos méthodes donnent de meilleurs résultats en termes de qualité de partitionnement que le LPA. Cependant, leur complexité est plus élevée.

Au chapitre 4, nous proposons une version parallèle et distribuée du CDLP pour travailler sur les grands graphes ayant plusieurs millions d'arêtes.

3.2 Propagation de labels avec détection de cœurs, ordre et coloration pour la détection de communautés disjointes

Nous avons utilisé dans la section précédente la notion de barrages en utilisant la centralité d'intermédiarité fondée sur les arêtes. Bien que permettant de meilleurs résultats que le LPA, la complexité algorithmique pour le calcul de la centralité d'intermédiarité est de $\mathcal{O}(n^3)$ (et peut être réduite en $\mathcal{O}(nm)$ par approximation). Nous nous proposons d'observer le comportement de la propagation de labels avec cœurs en utilisant un ordre de visite fondé sur certaines mesures sociales. L'objectif est d'étudier à la fois la stabilité de l'algorithme et la qualité par rapport aux versions précédemment proposées.

Nous avons vu au chapitre 1 que la propagation de labels avait été modélisée en utilisant la coloration dans un but de parallélisation, nommée la propagation de labels semi-synchrone. Ainsi, des groupes de nœuds connectés ayant la même coloration peuvent faire la mise à jour de leurs labels alors que d'autres sont dans un état d'attente (ou en "pause"). En itérant sur toutes les couleurs, une propagation de label globale a lieu. L'idée est de réutiliser le concept de coloration mais pour effectuer une mise à jour des labels des nœuds en fonction de leurs structures sociales et topologiques au sein du réseau. En introduisant un ordre fondé sur la topologie des groupes de nœuds d'une coloration, la propagation de label pourra ainsi être effectuée. Il s'agit d'une proposition algorithmique dans un but expérimental, afin de savoir si l'ordre peut jouer sur la stabilité et sur la qualité de la détection de communautés.

3.2.1 Propagation de labels asynchrone avec détection de cœurs et coloration

En effectuant une coloration du graphe $G = (V, E)$, nous obtenons une partition $\mathcal{D} = \{D_1, D_2, ..., D_l\}$ (de l parties). L'idée est de trier ces groupes de nœuds non connectés en fonction de leurs caractéristiques topologiques. C'est alors qu'un ordre de visite permet de mettre à jour les labels des nœuds d'une certaine couleur de manière croissante ou décroissante selon les valeurs des résultats des mesures sociologiques utilisées.

Nous considérons comme ordre :
— la centralité de degrés
— l'aléatoire

En considérant l'exemple suivant, on peut s'apercevoir que l'ordre fondé sur la coloration permet de réduire le nombre de propagations de labels et de gagner

en temps de calcul.

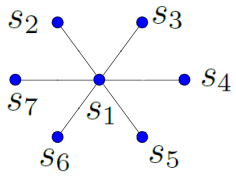


Figure 3.34 – Exemple de propagation de label semi-sychrone fondé sur l'ordre de visite

En appliquant un algorithme de coloration, nous obtenons les deux groupes de nœuds $D_1 = \{s_1\}$ et $D_2 = \{s_2, s_3, s_4, s_5, s_6, s_7\}$. Le degré moyen des nœuds de D_1 (ici juste du nœud s_1) est de 6 alors que le degré moyen du groupe D_2 est de 1. Notons par σ, la matrice qui représente l'ordre de visite pour la propagation de labels, où les éléments de la i^{eme} colonne représentent en première ligne le numéro de visite du nœud dont l'identifiant est en seconde ligne (s_i). En commençant par la mise à jour des labels des nœuds étant dans le groupe dont la centralité est la plus faible, nous obtenons l'ordre et la propagation de labels suivants :

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ s_2 & s_4 & s_5 & s_6 & s_7 & s_3 & s_1 \end{pmatrix}$$

$$s_2 \leftarrow s_1, s_4 \leftarrow s_1 \,, s_5 \leftarrow s_1, s_6 \leftarrow s_1, s_7 \leftarrow s_1, s_3 \leftarrow s_1, s_1 \leftarrow \{s_2, s_3, s_4, s_5, s_6, s_7\}$$

Une seule itération de propagation de labels suffit pour que tous les nœuds aient le même label. En ayant considéré l'ordre commençant par D_1 et ensuite D_2, nous aurions dû avoir plusieurs itérations de la propagation de label pour obtenir le même label pour tous les nœuds.

La complexité de l'algorithme dans le pire cas est en $\mathcal{O}(n \times \mathcal{N} \times k \times (m) + n^2 + n + m)$, où k est le nombre d'itérations de propagations de labels et m le nombre d'arêtes du graphe.

Dans le cadre de l'Algorithme 6, nous proposons de mettre à jour les labels des nœuds en commençant par les groupes de nœuds de même couleur ayant la centralité la plus faible (POP-UP), cela de manière croissante. Nous proposons également une seconde version, qui consiste à commencer de manière décroissante, des groupes de nœuds ayant la centralité la plus élevée vers ceux ayant la centralité la moins forte (POP-DOWN). Le cas où l'ordre est aléatoire sera noté R-POP (R pour random). De ces trois dernières propositions, il est possible, en faisant fluctuer la valeur de α, d'obtenir des dendrogrammes. La question est de savoir si elles sont foncièrement différentes, tant en termes de niveaux qu'en termes de partition.

Algorithme 6 La propagation de label semi-synchrone avec ordre préférentiel (POP)

Input : Un graphe $G = (V, E)$, un seuil α, $\mathcal{N}$ le nombre de propagations de labels, une mesure sociale $\mathbf{M}$ fondée sur les nœuds

Output : Les communautés trouvées par l'algorithme sur le graphe $G = (V, E)$

1: Effectuer une coloration du graphe G, $\mathcal{D} = \{D_1, D_2, ..., D_l\}$ (de l parties)
2: Allouer une matrice de fréquence vide $P_{ij}^{\mathcal{N}}$
3: Pour chaque groupe de nœuds d'une même couleur D_i, calculer la mesure moyenne de la mesure sociale $\mathbf{M}$, puis trier les groupes de nœuds suivant les résultats de manière croissante ou décroissante pour créer un ordre de visite σ.
4: Appliquer $\mathcal{N}$ fois la propagation de labels semi-synchrone en utilisant l'ordre préférentiel σ et remplir la matrice $P_{ij}^{\mathcal{N}}$
5: Créer un nouveau graphe $G' = (V, E')$ issu de $P_{ij}^{\mathcal{N}}$ dont les arêtes ont un poids supérieur ou égal à α
6: Créer une partition P en considérant les $\mathcal{C}$ composantes connexes comme cœurs
7: **Retourner** la partition $P = \{P_1, ..., P_{\mathcal{C}}\}$.

3.2.2 Expérimentation sur R-POP, POP-UP et POP-DOWN

Dans cette partie expérimentale, nous souhaitons :

1. savoir si l'ordre de visite fondé sur la centralité moyenne des groupes de nœuds ayant la même couleur améliorera la qualité des communautés détectées
2. connaître le nombre $\mathcal{N}$ de propagations de labels semi-synchrones à lancer pour stabiliser le processus de détection de communautés
3. savoir s'il existe un seuil pour lequel la valeur α n'est plus sujette à l'ordre de visite
4. savoir si l'ordre de visite a une conséquence sur la taille des dendrogrammes créés

Dans la suite des expérimentations, nous notons par I_x le fait que les propagations de labels ont été itérées x fois.

Etude sur la stabilisation de la propagation de label semi-synchrone

Les expérimentations portées sur trois réseaux montrent que l'ordre joue un rôle sur la stabilisation de la propagation de labels.

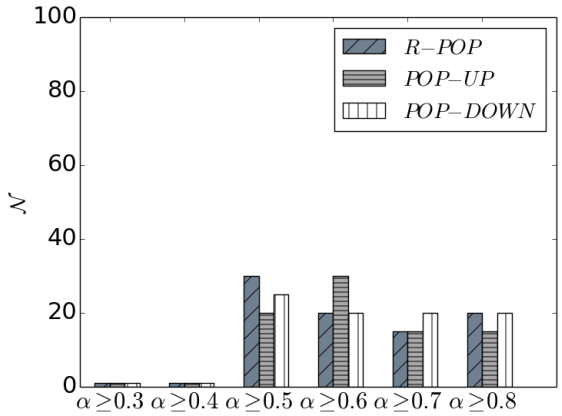


Figure 3.35 – Stabilité de la propagation de labels semi-sychrone fondée sur l'ordre de visite sur Zachary

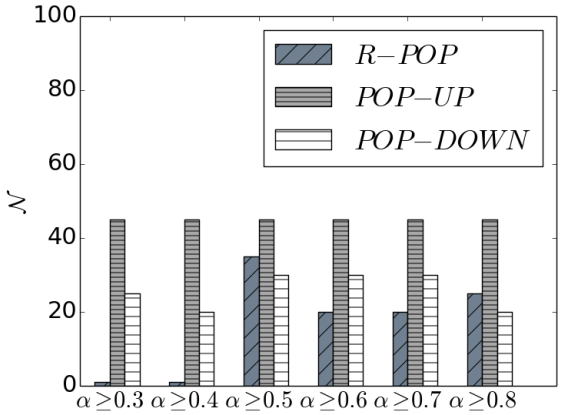


Figure 3.36 – Stabilité de la propagation de labels semi-sychrone fondée sur l'ordre de visite sur les dauphins

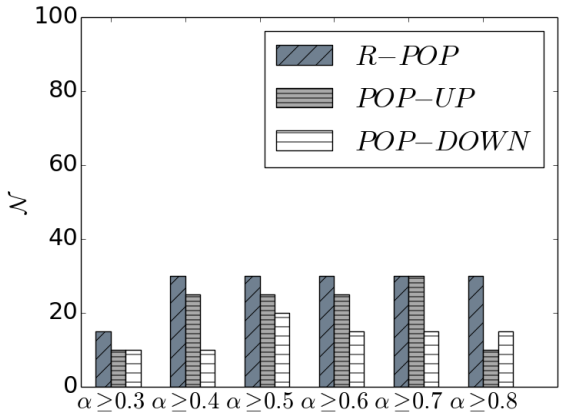


Figure 3.37 – Stabilité de la propagation de labels semi-sychrone fondée sur l'ordre de visite sur les livres politiques

En comparant la stabilisation du CDLP et des algorithmes à base de coloration, Figures 3.9, 3.35,3.36 et 3.37, on observe que les méthodes à base de coloration nécessitent moins de LPA pour arriver à stabilisation de l'algorithme pour des $\alpha \leq 0.8$. Nous observons que l'ordre joue un rôle également sur la stabilité de l'algorithme.

Pour Zachary, Figure 3.35, avec $\alpha \leq 0,5$, R-POP ne détecte pas de communauté alors que CDLP en détecte 2. Pour les algorithmes à base de coloration, il faut une vingtaine de LPA pour arriver à stabilisation. α a une incidence plus forte sur les algorithmes sans coloration que sur ceux avec.

Pour les dauphins, Figure 3.36, l'ordre a un rôle plus marqué sur le nombre de LPA nécessaires à la stabilisation. POP-UP nécessite entre 45 et 50 LPA contre une vingtaine pour POP-DOWN et POP-UP. Pour $\alpha \geq \{0.3; 0.4\}$ R-POP ne trouve pas de communautés. CDLP est plus sensible à α que les méthodes à base de coloration.

Pour les livres politiques, Figure 3.37, CDLP ne trouve pas de communauté pour $\alpha \geq \{0.3; 0.4\}$ alors que les autres méthodes détectent de petits groupes de nœuds dès $\alpha \geq 0.3$ qui correspondent aux livres neutres sur le plan politique. POP-DOWN est l'algorithme nécessitant le moins de LPA pour arriver à stabilisation.

Etude expérimentale sur des réseaux réels avec connaissance des communautés de terrains

Le Club de Karaté de Zachary

Coloration de graphe sur Zachary			
Couleurs	distribution	$\bar{d}$	Var
C_1	38.235 %	7.383	39.172
C_2	26.47 %	3.889	4.543
C_3	20.588 %	4.286	7.061
C_4	5.882 %	9.0	9.0
C_5	5.882 %	4.5	0.25
C_6	2.941 %	17.0	0.0

Tableau 3.5 – Distribution des tailles pour chaque groupe de nœuds ayant la même couleur C_i, $\bar{d}$ représente le degré moyen du groupe de nœuds ayant la couleur C_i et Var représente la variance concernant le degré des nœuds au sein d'une même couleur.

R-POP sur Zachary (I_5 ey $\mathcal{N} = 100$)					
	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	$\alpha \geq 0.9$
Modularité	0.0	0.133	0.399	0.376	0.376
Conductance	0.0	0.001	0.032	0.061	0.061
NMI	0.0	0.206	0.649	0.615	0.615
ARI	0.0	0.072	0.63	0.593	0.593
Pureté	0.5	0.647	0.97	0.97	0.97
#	1	2	4	5	5

Tableau 3.6 – Mesures supervisées et non supervisées avec R-POP

POP-UP sur Zachary (I_5 et $\mathcal{N} = 100$) OK					
	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	$\alpha \geq 0.9$
Modularité	0.37	0.399	0.399	0.376	0.392
Conductance	0.0004	0.0023	0.002	0.032	0.035
NMI	0.837	0.691	0.691	0.653	0.571
ARI	0.882	0.684	0.684	0.648	0.472
Pureté	0.97	0.97	0.97	0.97	0.97
#	2	3	3	4	5

Tableau 3.7 – Mesures supervisées et non supervisées avec POP-UP

POP-DOWN sur Zachary (I_5 et $\mathcal{N} = 100$) OK					
	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	$\alpha \geq 0.9$
Modularity	0.372	0.402	0.402	0.402	0.376
Conductance	0.0004	0.002	0.002	0.002	0.061
NMI	0.677	0.568	0.568	0.568	0.615
ARI	0.772	0.59	0.59	0.59	0.593
Pureté	0.941	0.941	0.941	0.941	0.97
#	2	3	3	3	5

Tableau 3.8 – Mesures supervisées et non supervisées avec POP-DOWN

Sur la figure 3.38, nous exposons les résultats de R-POP, POP-UP et POP-DOWN en lançant 100 propagations de labels ($\mathcal{N} = 100$). Les nœuds 0 et 33 représentent respectivement le manager et l'entraîneur qui se sont fâchés. D'après les Tableaux 3.7, 3.6 et 3.8, et leurs représentations sur les Figures 3.41, 3.39 et 3.40 avec la visualisation sur la figure 3.38, nous obtenons 6 niveaux pour le dendrogramme avec R-POP et 5 niveaux pour POP-UP et POP-DOWN. POP-UP a un NMI de 0.837 pour $\alpha \geq 0.5$ et produit les meilleurs résultats alors que POP-DOWN produit les moins bons. POP-UP et POP-DOWN trouvent bien les deux communautés pour $\alpha \geq 0.5$ alors que R-POP ne trouve qu'une seule grande communauté. Pour $\alpha \geq 0.6$ R-POP trouve une petite communauté reliée au manager, alors que 3 communautés sont détectées par les autres méthodes. Pour R-POP et POP-DOWN, les meilleurs résultats sont obtenus pour $\alpha \geq 0.7$. Pour $\alpha \geq 0.8$, POP-DOWN conserve une conductance assez faible alors qu'elle devient importante pour R-POP et POP-UP. Au niveau du nombre de communautés, nous obtenons un nombre très semblable pour toutes les méthodes. Les changements se font vis-à-vis de nœuds particuliers souvent considérés comme chevauchants dans littérature. On note que le nœud "9", qui est souvent considéré comme appartenant à deux communautés est détecté seul avec R-POP, alors qu'avec POP-UP, il rejoint la plus grande communauté et avec POP-DOWN, la plus petite.

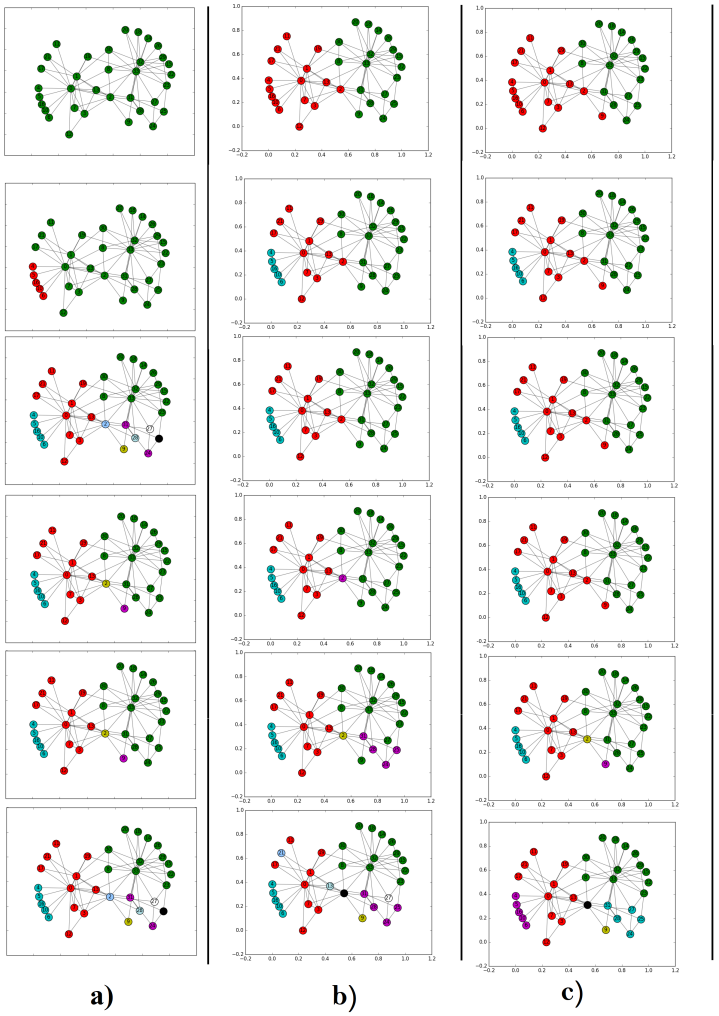


Figure 3.38 – Dendrogramme (Etude visuelle) sur Zachary, a) R-POP b) POP-UP c) POP-DOWN d)les valeurs de α (supérieur strictement)

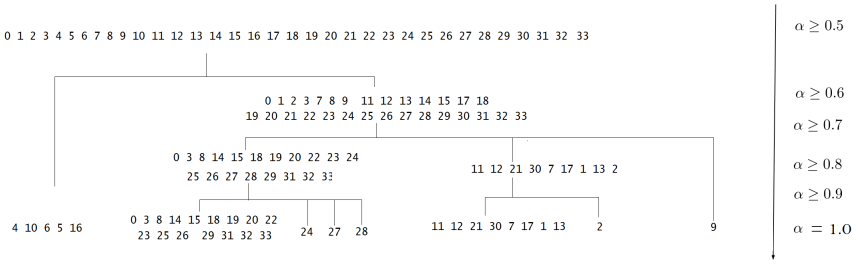


Figure 3.39 – Dendrogramme avec R-POP sur Zachary

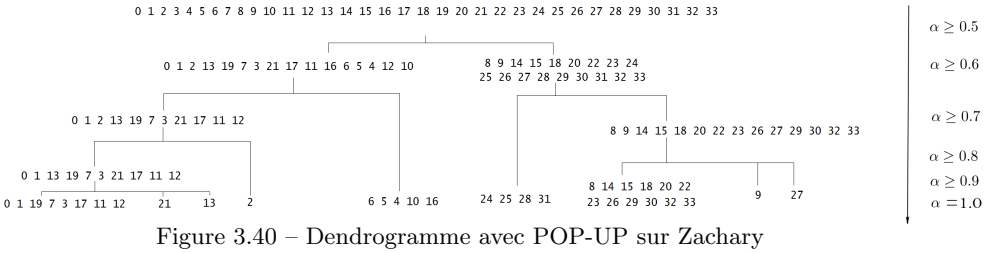


Figure 3.40 – Dendrogramme avec POP-UP sur Zachary

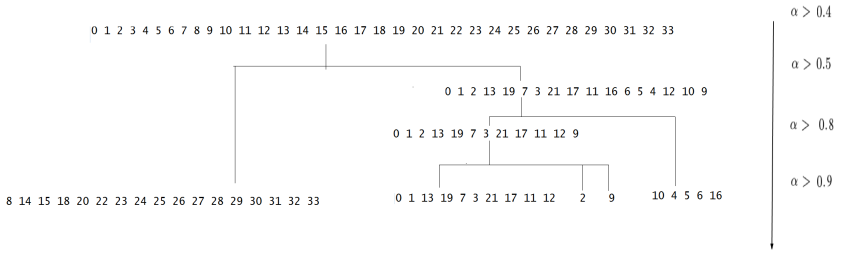


Figure 3.41 – Dendrogramme avec POP-DOWN sur Zachary

Coloration de graphe sur les dauphins			
Couleurs	distribution	$\bar{d}$	Var
C_1	40.322 %	4.884	4.289
C_2	22.58 %	4.857	7.979
C_3	14.516 %	5.889	6.321
C_4	11.29 %	7.571	4.531
C_5	6.452 %	8.0	3.5
C_6	3.226 %	8.0	1.0
C_7	1.613 %	10.0	0.0

Tableau 3.9 – Distribution des tailles pour chaque groupe de nœuds ayant la même couleur C_i, $\bar{d}$ représente le degré moyen du groupe de nœud ayant la couleur C_i et Var représente la variance concernant le degré des nœuds au sein d'une même couleur.

R-POP sur les Dauphins (I_5 and $\mathcal{N} = 100$)					
	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	$\alpha \geq 0.9$
Modularité	0.478	0.51	0.498	0.492	0.461
Conductance	0.0004	0.018	0.0503	0.07	0.082
NMI	0.756	0.638	0.604	0.5555	0.476
ARI	0.63	0.469	0.443	0.386	0.272
Pureté	1.0	1.0	1.0	1.0	1.0
#	3	5	7	9	12

Tableau 3.10 – Mesures supervisées et non supervisées avec R-POP

Nous obtenons 7 couleurs 3.9. la distribution n'est pas uniforme, avec les

POP-UP sur les dauphins (I_5 and $\mathcal{N} = 100$)					
	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	$\alpha \geq 0.9$
Modularité	0.48	0.5202	0.52	0.483	0.461
Conductance	0.0004	0.0013	0.001	0.071	0.082
NMI	0.676	0.585	0.5852	0.506	0.476
ARI	0.58	0.43250	0.4325	0.295	0.272
Pureté	0.984	0.984	0.984	1.0	1.0
#	3	4	4	10	12

Tableau 3.11 – Mesures supervisées et non supervisées avec POP-UP

POP-DOWN sur les dauphins (I_5 and $\mathcal{N} = 100$)				
	$\alpha \geq \{0.3, 0.4, 0.5\}$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$
Modularité	0.373	0.5196	0.515	0.505
Conductance	6.2*e-05	0.001	0.0208	0.0372
NMI	1.0	0.652	0.594	0.582
ARI	1.0	0.465	0.412	0.408
Pureté	1.0	1.0	1.0	1.0
#	2	4	6	7

Tableau 3.12 – Mesures supervisées et non supervisées avec POP-DOWN

deux couleurs les plus importantes représentant à elles seules 62.902 % de la population totale (en termes de nœuds). D'après les résultats des tableaux 3.10, 3.11 et 3.12, le dendrogramme de R-POP possède 6 niveaux, comme pour POP-DOWN. POP-DOWN trouve parfaitement les deux communautés avec un NMI de 1.0 avec de faibles valeurs de α ($\{0.3; 0.2\}$). Pour POP-UP, nous obtenons 5 niveaux. En termes de qualité, c'est POP-DOWN qui produit les meilleurs résultats et qui est persistant par rapport à α. La qualité se détériore plus rapidement avec POP-UP et R-POP lorsque α augmente.

Le réseau footballistique de Newman

Coloration de graphe sur le réseau footballistique			
Couleur	distribution	d	Var
C_1	16.522 %	10.853	1.008
C_2	14.783 %	10.705	1.031
C_3	16.522 %	10.579	0.875
C_4	15.652 %	10.5	0.6944
C_5	12.174 %	11.0	0.143
C_6	11.304 %	10.538	0.71
C_7	6.956 %	10.5	0.25
C_8	4.348 %	10.6	0.64
C_9	1.739 %	11.0	0.0

Tableau 3.13 – Distribution des tailles pour chaque groupe de nœuds ayant la même couleur C_i, $\bar{d}$ représente le degré moyen du groupe de nœuds ayant la couleur C_i et Var représente la variance concernant le degré des nœuds au sein d'une même couleur.

R-POP sur le réseau footballistique (I_5 et $\mathcal{N} = 100$)					
	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	$\alpha \geq 0.9$
Modularité	0.053	0.284	0.134	0.102	0.0628
Conductance	0.002	0.055	0.192	0.333	0.51
NMI	0.172	0.575	0.663	0.6832	0.699
ARI	0.015	0.2927	0.269	0.2362	0.164
Pureté	0.2	0.591	0.722	0.8	0.904
#	3	12	18	40	55

Tableau 3.14 – Mesures supervisées et non supervisées avec R-POP

POP-UP sur le réseau footballistique (I_5 et $\mathcal{N} = 100$)					
	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	$\alpha \geq 0.9$
Modularity	0.478	0.51	0.498	0.492	0.375
Conductance	0.0004	0.018	0.05	0.07	0.249
NMI	0.756	0.638	0.604	0.555	0.417
ARI	0.63	0.469	0.443	0.386	0.214
Pureté	1.0	1.0	1.0	1.0	1.0
#	3	5	7	9	21

Tableau 3.15 – Mesures supervisées et non supervisées avec POP-UP

POP-DOWN sur Football (I_5 et $\mathcal{N} = 100$)					
	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	$\alpha \geq 0.9$
Modularité	0.148	0.242	0.11	0.054	0.033
Conductance	0.002	0.087	0.222	0.451	0.607
NMI	0.248	0.545	0.628	0.655	0.693
ARI	0.046	0.197	0.221	0.116	0.109
Pureté	0.235	0.548	0.704	0.817	0.922
#	3	13	17	42	67

Tableau 3.16 – Mesures supervisées et non supervisées avec POP-DOWN

L'algorithme de coloration trouve bien 9 couleurs dont la distribution sur les nœuds est assez uniforme. 6 couleurs représentent près de 86,99 % de nœuds. Les résultats sur cet exemple sont assez différents, notamment en fonction de α. Pour R-POP et POP-DOWN, les meilleurs résultats sont obtenus pour $\alpha \geq 0.6$. Cependant, le nombre de communautés résultantes par ces méthodes est très volatil par rapport à α. Pour R-POP et POP-DOWN, après $\alpha \geq 0.7$, le nombre de communautés augmente considérablement.

Les livres politiques de Krebs (élection présidentielle américaine de 2004)

Coloration de graphes sur les livres politiques de Krebs			
Couleurs	distribution	d	Var
C1	30.476 %	8.41	45.05
C2	22.857 %	8.0	21.5
C3	15.238 %	8.75	33.81
C4	10.476 %	9.091	34.44
C5	11.428 %	9.25	18.18
C6	3.809 %	15.5	41.25
C7	3.809 %	11.5	17.25
C8	1.904 %	20.5	6.25

Tableau 3.17 – Distribution des tailles pour chaque groupe de nœuds ayant la même couleur C_i, $\bar{d}$ représente le degré moyen du groupe de nœud ayant la couleur C_i et Var représente la variance concernant le degré des nœuds au sein d'une même couleur.

R-POP sur les livres politiques de Krebs (I_5 et $\mathcal{N} = 100$)					
	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	$\alpha \geq 0.9$
Modularité	0.493	0.493	0.503	0.503	0.485
Conductance	0.0002	0.0002	0.03	0.03	0.04
NMI	0.578	0.578	0.55	0.550	0.515
ARI	0.69	0.69	0.668	0.668	0.55
Pureté	0.85	0.847	0.857	0.857	0.867
#	3	3	7	7	9

Tableau 3.18 – Mesures supervisées et non supervisées avec R-POP

POP-UP sur les livres politiques de Krebs (I_5 et $\mathcal{N} = 100$)					
	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	$\alpha \geq 0.9$
Modularité	0.09	0.0905	0.494	0.494	0.483
Conductance	0.0002	0.0002	0.01	0.01	0.047
NMI	0.049	0.049	0.592	0.592	0.504
ARI	0.057	0.057	0.703	0.703	0.521
Pureté	0.486	0.486	0.857	0.857	0.867
#	2	2	4	4	10

Tableau 3.19 – Mesures supervisées et non supervisées avec POP-UP

La Figure 3.42 et les tableaux 3.18, 3.19 et 3.20 sont la retranscription des résultats de R-POP, POP-DOWN et POP-UP. Pour $\alpha \geq 0.5$, R-POP trouve bien 3 communautés, avec pour chaque communauté, une identification politique

POP-DOWN sur les livres politiques de Krebs (I_5 et $\mathcal{N} = 100$)					
	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	$\alpha \geq 0.9$
Modularité	0.457	0.4568	0.489	0.489	0.49
Conductance	0.000	0.000	0.011	0.011	0.046
NMI	0.598	0.598	0.585	0.585	0.539
ARI	0.667	0.667	0.705	0.705	0.668
Pureté	0.848	0.848	0.866	0.866	0.876
#	2	2	5	5	10

Tableau 3.20 – Mesures supervisées et non supervisées avec POP-DOWN

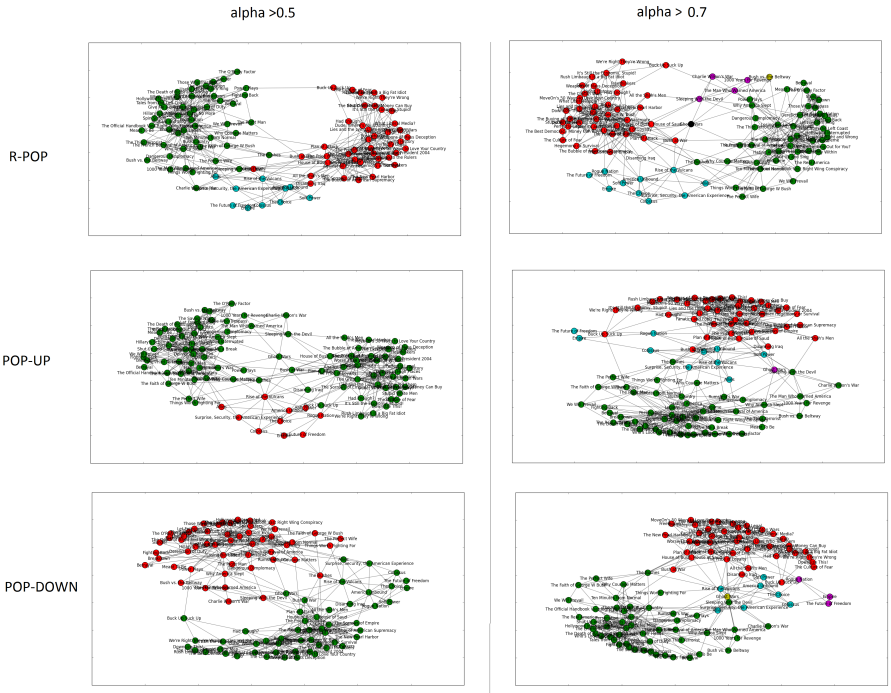


Figure 3.42 – **Visualisation des communautés pour** $\alpha \geq 0.5$ **et** $\alpha \geq 0.7$ **a) R-POP b) POP-UP c) POP-DOWN**

distincte. Nous notons que la communauté représentant les livres "neutres" sur le plan politique est dispatchée entre les trois principales communautés. POP-DOWN ne détecte que deux communautés, les républicains et les démocrates, les neutres étant majoritairement placés dans la communauté démocrate. Pour $\alpha \geq 0.7$, R-POP est celui qui dissocie le mieux les neutres des deux plus grands partis. Les démocrates et les républicains sont parfaitement détectés.Une partie des neutres se retrouve néanmoins chez les démocrates. POP-UP détecte la majorité des neutres, mais la coupe en deux grandes parties.

D'après les résultats, les principaux groupes de nœuds, à savoir les démocrates

et les républicains sont bien détectés par les trois méthodes. Les différences portent entre les livres neutres sur le plan politique.

3.2.3 Conclusion sur les propositions algorithmiques à base de coloration

L'idée de donner un ordre de visite en utilisant la coloration fut d'éviter la survenue de mauvaises propagations, c'est-à-dire, d'éviter qu'une propagation suive un chemin menant à une communauté géante par succession de choix aléatoires inadéquats.

Nos expérimentations ont montré que l'ordre de visite a un impact sur la qualité de partitionnement, mais également vis-à-vis du seuil α lors des détections des toutes premières communautés. Par exemple, avec le réseau de dauphins, R-POP ne détecte les premières communautés qu'à partir de $\alpha \geq 0.5$ alors que POP-UP et POP-DOWN ne détectent leurs premières communautés qu'à partir de $\alpha \geq 0.3$. Le nombre de communautés varie selon les méthodes. Par exemple, sur le réseau footballistique, avec $\alpha \geq 0.7$, POP-UP détecte 7 communautés alors que R-POP en détecte 11 et POP-DOWN en détecte 17.

Nos expérimentations ne démontrent pas qu'un ordre déterminé améliore la qualité de partitionnement mais révèlent l'obtention de partitions de qualité différente. Cependant, ces méthodes sont déterministes. Nous verrons qu'il est délicat de comparer les résultats de qualité de partitionnement avec les autres algorithmes car l'ordre induit une incertitude quant aux partitions obtenues.

3.3 Analyse comparative des méthodes disjointes

Nous proposons d'utiliser les algorithmes de la littérature suivants pour la détection de communautés, à savoir : la méthode de Girvan et Newman (**GN**) Girvan et Newman (2002a), la méthode de Louvain (**Louvain**) Blondel *et al.* (2008) et sa version avec les cœurs Seifi *et al.* (2013), la méthode spectrale fondée sur l'optimisation de la modularité (**spectral Newman**) de Newman (2006), le modèle de Potts (**spin**) de Ronhovde et Nussinov (2010), le LPA (**LPA**) de Raghavan *et al.* (2007) et ses améliorations comme Šubelj et Bajec (2011) (**DPA**) et Leung *et al.* (2009) (**Leung**) , l'algorithme à base de leaders (**LICOD**) de Kanawati (2011), un algorithme venant de la théorie de l'information (**Infomap**) de Rosvall et Bergstrom (2007a), et finalement, la méthode de marche aléatoire (**Walktrap**) de Pons et Latapy (2006).

Les observations des tableaux 3.23, 3.21 et 3.22 montrent que les algorithmes MPLBS et PLAB proposés apportent des améliorations par rapport à ceux issus de la littérature, donnant de bonnes valeurs pour le NMI et le ARI, ainsi que notre version finale PLBS. En effet, PLBS pour le réseau de football obtient un NMI de 0.931 et un ARI de 0.907.
Nos expérimentations ont également montré qu'alimenter une matrice d'appartenance avec différents barrages pouvait donner de très bons résultats comme le montre PLBS. En effet, selon les résultats des mesures supervisées, PLBS montre qu'elle donne de meilleurs résultats dans la majeure partie des cas par rapport au MPLBS. Imposer des barrages limite la propagation de labels et évite le fait de tomber dans de trop grandes communautés. C'est ce que montrent les résultats de PLAB qui nécessitent cependant de tester β. Pour la paramétrisation, nous avons choisi $\alpha = 0.5$ pour tous les réseaux de PLBS. Un α trop fort donnerait de trop nombreuses communautés, et trop faible, risquerait de ne donner qu'une communauté. Le MPLBS utilisant la modularité, nous retourne la paramétrisation maximisant cette dernière, avec la partition résultante. Une dernière observation montre que PLBS et MPLBS donnent des conductances assez faibles en comparaison des autres algorithmes de la littérature, ceci s'expliquant par le fort nombre de barrages. Par rapport au LPA ou à la méthode de Louvain, nos méthodes sont déterministes.

Concernant les algorithmes à base de colorations et de visites préférentielles, nos algorithmes donnent des résultats très satisfaisants par rapport à la littérature. On observe que l'ordre joue un rôle sur le déterminisme de la méthode utilisant la matrice de fréquence. On a pu observer que les communautés classiques avaient été détectées par toutes les méthodes (R-POP,POP-UP et POP-DOWN) mais que la différence résidait surtout sur certains nœuds qui sont d'après la littérature chevauchants. Comme l'exemple de Zachary ou celui des livres politiques, les nœuds qui diffèrent de communautés selon la méthode utilisée sont surtout des nœuds appelés chevauchants. C'est-à-dire que ce sont des nœuds susceptibles d'appartenir à plusieurs communautés. On observe également qu'en

Analyse comparative avec certains algorithmes de détection de communautés						
	Q	Φ	NMI	ARI	Pureté	#
Zachary #2						
Louvain	**0.420**	0.005	0.49	0.392	0.941	4
Seifi	**0.420**	0.005	0.49	0.392	0.941	4
GN	0.401	0.0348	0.485	0.3915	0.941	5
Spin	0.005	0.2740	0.588	0.464	**0.97**	4
Spectral	0.393	0.005	0.579	0.435	**0.97**	4
WalkTrap	0.353	0.01	0.49	0.321	0.912	5
Leung	0.399	0.002	0.328	0.515	**0.97**	3
LPA*	0.336	**0.001**	0.575	0.570	0.895	2.61
DPA	**0.42**		0.660			
Infomap	0.402	0.002	0.568	0.59	0.941	3
LICOD	0.24		0.60	0.62		3
SLPH*	0.347	0.147	0.615	0.628	0.909	2
R-POP	0.399	0.032	0.649	0.63	**0.97**	5
POP-UP	0.399	0.002	**0.691**	**0.684**	**0.97**	3
POP-DOWN	0.402	0.002	0.568	0.59	0.941	3
PLBS	0.378	0.073	0.565	0.498	**0.97**	6
PLAB	0.399	0.178	**0.691**	**0.684**	**0.97**	3
MPLBS	0.402	0.181	0.5684	0.591	0.941	3
CDLP	0.376	0.242	0.6154	0.593	**0.97**	5
Football #11						
Louvain	0.602	**0.002**	0.855	0.728	0.922	9
Seifi	0.602	**0.002**	0.855	0.728	0.8	9
GN	0.600	0.003	0.879	0.778	0.835	10
Spin	0.003	0.293	0.892	0.816	0.8696	10
Spectral	0.488	0.002	0.695	0.891	0.626	8
WalkTrap	**0.604**	0.002	0.887	0.815	0.922	10
Leung	0.569	0.005	0.900	0.836	0.922	13
LPA*	0.595	0.005	0.887	0.779	0.854	10.39
DPA	0.606		0.9			
Infomap	0.579	0.013	0.9	0.853	0.913	12
LICOD	0.49		0.83	0.69		16
SLPH*	0.59	0.312	0.893	0.799	0.879	4
R-POP	0.284	0.055	0.555	0.293	0.591	18
POP-UP	0.510	0.018	0.638	0.469	**1.0**	5
POP-DOWN	0.242	0.087	0.545	0.197	0.548	21
PLBS	0.584	0.042	**0.931**	**0.907**	0.957	16
PLAB	0.598	0.317	0.929	0.900	0.939	13
MPLBS	0.602	0.310	0.927	0.889	0.930	12
CDLP	0.602	0.310	0.927	0.889	0.930	12

Tableau 3.21 – Analyse comparative avec certains algorithmes issus de la littérature (Part 1)

* signifie que nous avons lancé l'algorithme 100 fois et que nous avons retranscrit la moyenne des scores.

Analyse comparative avec certains algorithmes de détection de communautés						
	Q	Φ	NMI	ARI	Pureté	#
Dauphins #2						
Louvain	0.519	0.002	0.510	0.327	0.968	5
Seifi	0.519	0.002	0.511	0.327	0.968	5
GN	0.519	0.006	0.554	0.395	0.984	5
Spin	**0.529**	0.003	0.586	0.373	**1.0**	5
Spectral	0.491	0.002	0.449	0.283	0.952	5
WalkTrap	0.489	0.005	0.537	0.417	0.952	4
Leung	0.519	0.003	0.481	0.299	0.968	6
LPA*	0.483	**0.000**	0.623	0.502	0.979	3.85
DPA	**0.529**		0.774			
Infomap	0.519	0.008	0.481	0.291	0.968	6
LICOD	0.35		0.41	0.32		2
SLPH*	0.491	0.003	0.621	0.488	0.984	4
R-POP	0.511	0.018	0.632	0.469	**1.0**	5
POP-UP	0.520	0.001	0.585	0.433	0.984	4
POP-DOWN	0.373	**0.000**	**1.0**	**1.0**	**1.0**	2
PLBS	0.275	0.068	0.547	0.597	0.903	3
PLAB	0.376	0.055	0.943	0.956	**1.0**	3
MPLBS	0.457	0.043	0.598	0.667	0.848	2
CDLP	0.518	0.197	0.631	0.454	**1.0**	5
Politiques #3						
Louvain	0.521	0.001	0.512	0.558	0.724	4
Seifi	0.521	0.001	0.512	0.558	0.848	4
GN	0.517	0.002	0.559	0.682	0.857	5
Spin	0.526	0.003	0.466	0.466	0.439	6
Spectral	0.467	0.001	0.520	0.547	0.847	4
WalkTrap	0.507	0.001	0.543	0.653	0.848	4
Leung	0.498	0.001	0.348	0.393	0.867	7
LPA*	0.492	**0.000**	0.555	0.651	0.845	3.38
DPA	**0.527**		**0.805**			
Infomap	0.522	0.002	0.493	0.536	0.848	6
LICOD	0.42		0.68	0.67		6
SLPH*	0.495	0.094	0.553	0.656	0.846	3
R-POP	0.503	0.030	0.551	0.668	0.857	7
POP-UP	0.495	0.009	0.592	**0.703**	0.857	4
POP-DOWN	0.49	0.046	0.539	0.668	0.876	10
PLBS	0.459	0.104	0.601	0.668	**0.933**	16
PLAB	0.495	0.078	0.573	0.675	0.848	4
MPLBS	0.495	**0.069**	0.586	0.689	0.848	3
CDLP	0.506	0.120	0.541	0.652	0.857	7

Tableau 3.22 – Analyse comparative avec certains algorithmes issus de la littérature (Part 2)

* signifie que nous avons lancé l'algorithme 100 fois et que nous avons pris la moyenne des scores.

Foot #11	PLAB	PLBS	MPLBS	*Dauphins* #2	PLAB	PLBS	MPLBS
α	$\{0.6\}$	$\{0.5\}$	$\{0.5\}*$	α	$\{0.6\}$	$\{0.5\}$	$\{0.3-0.5\}*$
β	$\{0.05\}$	$\Delta_{[0.3;0.6]}$	$\{0.3\}*$	β	$\{0.05\}$	$\Delta_{[0.0;0.3]}$	$\{0.3\}*$
Zac #2	PLAB	PLBS	MPLBS	*Pol* #3	PLAB	PLBS	MPLBS
α	$\{0.6\}$	$\{0.5\}$	$\{0.5\}*$	α	$\{0.6\}$	$\{0.5\}$	$\{0.3-0.5\}*$
β	$\{0.1\}$	$\Delta_{[0.3;0.6]}$	$\{0.5\}*$	β	$\{0.3\}$	$\Delta_{[0.3;0.6]}$	$\{0.0\}*$

Tableau 3.23 – Paramétrisation de nos algorithmes, "*" signifie que c'est le résultat retourné par l'algorithme par optimisation de la modularité. α est la valeur pour la création du graphe issu de la matrice de fréquence et β, le pourcentage de barrages.

moyenne, la méthode à base de coloration produit en général plus de communautés que les autres méthodes, avec une conductance plus importante, c'est-à-dire la création de très petites communautés.

3.4 Propagation de labels avec détection de cœurs pour la détection de communautés chevauchantes

La section précédente a présenté divers algorithmes pour la détection de communautés disjointes dont les objectifs sont d'améliorer le LPA. En considérant l'exemple du CDLP et par observation de la matrice de fréquence, on voit des nœuds fréquemment ensemble au cours des différentes LPA, mais aussi des nœuds induire une idée de chevauchement car appartenant à différentes communautés. Dans certains réseaux, comme celui de collaborations scientifiques où des chercheurs publient leurs travaux de recherche, il peut y avoir des personnes appartenant à plusieurs communautés. C'est par exemple le cas si un chercheur a publié un article dans le domaine des graphes, et un autre dans le domaine de l'oncologie. Les citations du premier se feront par des articles liés au graphe alors que le second sera cité par des articles traitant de l'oncologie. L'auteur sera donc cité par deux groupes de personnes dont les champs scientifiques sont différents. Nous proposons dans cette section la possibilité pour un nœud d'appartenir à plusieurs communautés suivant certaines conditions en utilisant une fonction d'appartenance fondée sur la topologie du graphe, la pondération des arêtes et les caractéristiques sociales des communautés.

3.4.1 De la propagation de labels avec détection de cœurs au chevauchement

L'objectif principal du CDLP présenté à la section précédente fut de stabiliser la propagation de labels en lançant plusieurs fois l'algorithme tout en utilisant une matrice de fréquence. Cependant, bien que cette méthode puisse mettre en exergue des cœurs de communautés, c'est-à-dire des nœuds fréquemment ensemble durant les différentes propagations de labels, elle montre que certains nœuds peuvent appartenir à des communautés différentes suite à l'application de plusieurs CDLP. Ce sont sur ces nœuds, dont l'imprécision d'appartenance à plusieurs classes est élevée, que nous portons notre attention. Dans le cadre de ce travail doctoral, nous proposons d'établir une mesure de décision permettant de spécifier pour un certain nombre de nœuds, délicats à classer, leurs différentes communautés d'appartenance possibles, créant par voie de conséquence, le chevauchement entre communautés. Pour ce faire, nous proposons de donner ou de rappeler certaines définitions que nous utiliserons pour l'élaboration d'une fonction de décision pour le chevauchement.

Définition Un cœur de communauté est l'ensemble des nœuds se trouvant fréquemment ensemble dans une même communauté après plusieurs lancements d'un algorithme non déterministe (exemple de Louvain avec Seifi *et al.* (2013)).

Ainsi, les nœuds étant presque toujours dans les mêmes communautés au cours des différentes propagations de labels ne portent pas à confusion quant à

la communauté à laquelle ils appartiennent. Cependant, certains nœuds ont une fréquence d'apparition vis-à-vis de certaines communautés bien plus faible. Nous portons notre attention sur les nœuds qui sont liés à plusieurs communautés. Nous définissons ainsi la bordure de communauté.

Définition La bordure (ou frontière) d'une communauté est l'ensemble des nœuds ayant au moins un lien vers une autre communauté. Elle constitue l'ensemble des *candidats potentiels au chevauchement*.

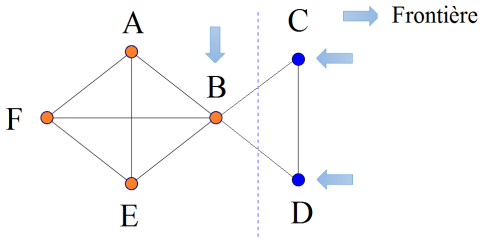


Figure 3.43 – Exemple de bordure (ou frontière), ici les nœuds B, C et D

Dans l'exemple de la Figure 3.43, il y a deux communautés avec la partition $P = \{\{A, B, E, F\}, \{C, D\}\}$. En considérant la communauté $P_1 = \{A, B, E, F\}$, seul le nœud B possède des liens à l'extérieure de la sa communauté, il appartient donc à la bordure de sa communauté (sa frontière). En considérant la communauté $P_2 = \{C, D\}$, les nœuds C et D possèdent au moins un lien vers une autre communauté, ils sont donc aussi sur la bordure de leurs communautés.

Pour nos études expérimentales, nous utilisons certaines statistiques portant sur les nœuds chevauchants. Nous définissons pour cela *le pourcentage de chevauchement*, *le taux de chevauchement pour un nœud* et *le taux de chevauchement moyen*.

Définition Le pourcentage de chevauchement est le pourcentage de nœuds appartenant à plusieurs communautés.

Définition Le taux de chevauchement pour un nœud est le rapport entre le nombre de communautés auxquelles le nœud appartient et le nombre de communautés auxquelles il est relié.

Définition Le taux de chevauchement moyen est la moyenne des taux de chevauchement de tous les nœuds du graphe.

3.4.2 Fonction d'appartenance

La proposition algorithmique pour la détection de communautés chevauchantes est fondée sur l'aspect sociologique d'un individu vis-à-vis de ses relations et des communautés auxquelles il est lié.

Considérons l'exemple sociologique suivant où le nœud étudié est u avec deux communautés d'individus. Les nœuds sont des individus et les arêtes leurs degrés d'amitiés. On suppose que le nœud u est ami avec Rebecca et avec Valentin. Lors d'une soirée, la question est de savoir vers quelle communauté pourrait aller le nœud u.

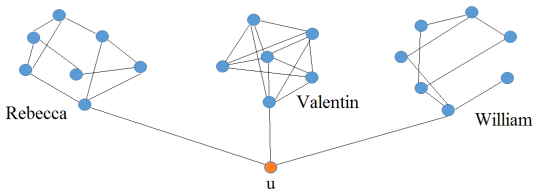


Figure 3.44 – Expérience sociale

Le nœud u a donc le choix entre trois communautés. Sa décision se fera selon son degré d'amitié avec Rebecca, Valentin et William mais aussi vis-à-vis des relations de ses amis. On peut s'apercevoir de la forte collaboration au sein du groupe de Valentin avec les nœuds ayant une forte centralité (certains ayant un degré de 4 ou de 5). Dans le groupe de Rebecca, la collaboration semble moins forte mais est présente. Dans le groupe de William, la collaboration semble faible, certains nœuds ne communiquent pas ensemble. En observant la topologie des structures communautaires, le nœud u pourrait rejoindre au cours de la soirée le groupe de Rebecca ou celui de Valentin. Cependant, le nœud u ira cotoyer un groupe si son lien avec l'individu est assez fort. Si u a une plus grande affinité pour Rebecca et Valentin que pour William, u ira cotoyer en premier le groupe de Rebecca puis celui de Valentin. Le nœud u pourrait alors appartenir à deux communautés. Notre proposition algorithmique repose sur le graphe seuillé G', résultante de la matrice de fréquence $P_{ij}^{\mathcal{N}}$. L'idée est de pouvoir utiliser la pondération des liens que l'on peut trouver dans la matrice de fréquence comme une relation d'amitié.

Le nouveau graphe G' est alors projeté sur le graphe originel G, tout en respectant sa topologie. Cela signifie que les arêtes présentes dans G' mais pas dans G ne seront pas considérées, afin de ne pas mettre de liens absurdes. Ce peut être le cas s'il y a des conflits entre personnes ou que des individus refusent de parler à d'autres individus, comme dans le graphe de Zachary entre le manager et l'entraîneur. Nous utilisons ainsi l'information stockée dans $P_{ij}^{\mathcal{N}}$ pour pondérer G. Cela nous permet de voir les paires de nœuds ayant une forte probabilité d'être ensemble en terme communautaire. En utilisant le seuil α sur le nouveau graphe pondéré, nous obtenons les communautés et les liens entre communautés (LEC). Les nœuds qui sont à la frontière de leurs communautés et reliés à d'autres sont de possibles candidats pour le chevauchement. Pour savoir si ces nœuds candidats sont de potentiels futurs nœuds chevauchants, nous proposons les fonctions d'appartenances suivantes.

Considérant un nœud candidat u, l'idée est de mesurer le pouvoir d'appartenance qu'a ce nœud en observant ses communautés avoisinantes et leurs structures topologiques. Nous écrivons $\mathcal{C}_u = \{\mathcal{C}_1^u, ..., \mathcal{C}_K^u\}$, les différentes commu-

nautés auxquelles le nœud u est lié. Si u est lié à K différentes communautés, nous le notons par $|\mathcal{C}_u| = K$.

Fonctions d'appartenance pour le chevauchement

En considérant les K différentes communautés présentes dans le voisinage du nœud u, nous proposons *la fonction d'appartenance* :

$$f_X : u \times \{C_1^u, ..., C_K^u\} \longmapsto \mathbb{R}_+ \tag{3.3}$$

$$f_X(u, \{C_1^u, ..., C_K^u\}) = \max_{c \in \binom{\mathcal{C}_K^u}{j}, j \in \{1,...,K\}} (\frac{1}{|c|} \times \sum_{i \in c} \omega_{u,i} X^S(i)) \tag{3.4}$$

où c est une liste de combinaisons de communautés voisines d'un nœud u et X un paramètre portant sur une mesure sociale fondée sur la structure topologique d'un sous-graphe. Le terme c va prendre toutes les combinaions possibles, notamment avec $j \in \{1, ..., K\}$. Par la suite, le terme $i \in c$ signifie que l'on va regarder toutes les combinaisons de communautés de c, ainsi, nous pourrons avoir les pondérations des liens $\omega_{u,i}$, où $\omega_{u,i}$ est la somme des poids des arêtes liant le nœud u aux communautés dans la liste i. Le terme $X^S(i)$ représente la valeur d'une mesure sociale du sous graphe (ici la communauté) S constitué de l'union des communautés de i.

Le cœfficient binomial $\binom{\mathcal{C}_K^u}{j}$ permet de calculer les j combinaisons dans un ensemble de $\mathcal{C}_K^u$ éléments, les éléments étant les communautés. Ainsi, un nœud va essayer toutes les combinaisons de communautés auquel il pourrait appartenir ($c \in \binom{\mathcal{C}_K^u}{j}$). Cela permet au nœud u d'appartenir à une ou plusieurs communautés. La configuration ayant le score le plus élevé permettra l'assignement du nœud u aux communautés choisies.

Nous notons dans la formule ci-dessus que $j \in \{1, ..., K\}$, mais il est tout à fait possible de forcer un nœud candidat soit chevauché par au moins L communautés, en écrivant $j \in \{L, ..., |C_K^u|\}$. Cela permet à l'utilisateur de choisir le nombre de communautés auxquelles peut appartenir un nœud candidat.

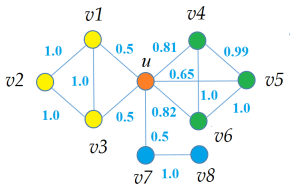
Pour les mesures sociales portant sur la structure topologique des communautés, nous proposons d'utiliser la densité (dont la fonction d'appartenance est noté f_d) et le cœfficient de clustering (dont la fonction d'appartenance est noté f_{cc}). Ces mesures sont utilisées pour effectuer une étude comparative.

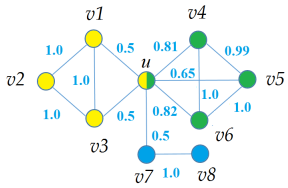
En ce qui concerne la fonction d'appartenance fondée sur la densité, un nœud avec une forte pondération sur ses liens connectés à des communautés ayant de fortes densités aura plus de chance d'être chevauchant plutôt qu'un nœud avec de faibles pondérations sur ses arêtes connectées à des communautés de faibles

densités. Cependant, dans certaines situations, les chevauchements ne peuvent pas se faire. Cela peut être le cas si un nœud u est très faiblement lié à ses communautés avoisinantes, elles mêmes de faibles densités, avec une valeur de f_d relativement faible. Fondé sur ce constat, le chevauchement sera effectué si et seulement si $f_d(u, \{C_1^u, ..., C_K^u\}) \geq \frac{1}{|c|} \sum_{S \in c} d^S$ où d^S est la densité du sous graphe (ici la communauté) S.

L'idée d'utiliser ensuite le cœfficient de clustering est que le nœud candidat observera les connexions au sein des communautés, et la présence de leaders, caractéristique des graphes de terrains dont la distribution des degrés des nœuds suit une loi faible. De même que pour la densité, si un nœud u est très faiblement lié à ses communautés avoisinantes, elles mêmes de faibles cœfficients de clustering, il n'y a pas de raison pour qu'il y ait de chevauchement. Fondé sur ce constat, le chevauchement sera effectué si et seulement si $f_{cc}(u, \{C_1^u, ..., C_K^u\}) \geq \frac{1}{|c|} \sum_{S \in c} cc^S$ où cc^S est le coefficient de clustering du sous graphe (ici la communauté) S.

Supposons que nous ayons le graphe de la *figure 3.45*, avec la partition $P = \{C_1, C_2, C_3, C_4\}$ telle que $C_1 = \{v_1, v_2, v_3\}$,$C_2 = \{v_4, v_5, v_6\}$, $C_3 = \{v_7, v_8\}$ and $C_4 = \{u\}$, résultante de la propagation de labels avec matrice de fréquence. La question est de savoir si le nœud u peut appartenir à plusieurs communautés. On observe que la meilleure configuration pour l'obtention de communautés chevauchantes sur le nœud u est donnée avec C_1 et C_2. Le nœud u est ainsi assigné dans la combinaison où les densités sont les plus élevées, et les pondérations des liens sont les plus fortes.





Expérimentations		
combinaisons	f_{cc} score	f_d score
$f_{cc}(u, \{C_1, C_2, C_3\})$	1.093	1.26
$f_{cc}(u, \{C_1, C_2\})$	**1.64**	**1.64**
$f_{cc}(u, \{C_1, C_3\})$	0.5	0.75
$f_{cc}(u, \{C_2, C_3\})$	1.14	1.39

Figure 3.45 – Après avoir calculé f_d et f_{cc} sur le nœud u, u appartient à deux communautés.

3.4.3 Propositions algorithmiques pour la détection de communautés chevauchantes

Nous exposons la propagation de labels avec détection de cœurs et chevauchement (CDLPOV), algorithme 7.

$f_{appartenance}$ fait référence à la fonction choisie par l'utilisateur, à savoir f_d ou f_{CC}. $SNM(S)$ est la mesure d'analyse des réseaux sociaux utilisée concernant l'aspect topologique de la communauté S, à savoir la densité ou le CC. En faisant

Algorithme 7 Le CDLP avec fonction d'appartenance (CDLPOV)

Entrée : Un graphe $G = (V, E)$, le seuil α, $\mathcal{N}$ le nombre de lancements
Sortie : Les communautés chevauchantes de G

1: Allouer une matrice de fréquence vide
2: Lancer $\mathcal{N}$ fois la propagation de labels asynchrone
3: Remplir la matrice de fréquence avec les résultats des $\mathcal{N}$ LPA
4: Créer un nouveau graphe $G' = (V, E')$ de $P_{ij}^{\mathcal{N}}$ avec des arêtes dont la pondération est supérieure ou égale à α
5: Projeter le graphe G' sur G avec la pondération (mais en enlevant les arêtes présentes dans G' mais pas dans G)
6: Créer une partition $P = \{P_1, ..., P_{\mathcal{C}}\}$ en considérant les $\mathcal{C}$ composantes connexes comme cœurs
7: Calculer les arêtes entre communautés (AEC)
8: $Cand \leftarrow []$ {$Cand$ est une liste de candidats potentiels au chevauchement (nœuds à la frontière des communautés)}
9: **Pour** chaque nœud u ayant une arête dans AEC **Faire**
10: $Cand$.ajouter(u)
11: **Fin Pour**
12: $P^{Ov} \leftarrow P$ {Partition chevauchante que renverra l'algorithme}
13: **Pour** chaque nœud u dans $Cand$ **Faire**
14: $Cand$.ajouter(u)
15: **Fin Pour**
16: **Pour** chaque nœud u dans $Cand$ **Faire**
17: **Si** $f_{appartenance}(u, \{C_1^u, ..., C_K^u\}) \geq \sum_{S \in \{C_1^u, ..., C_K^u\}} SNM(S)$ **Alors**
18: Dupliquer le nœud u dans les communautés correspondantes de P^{Ov}
19: **Fin Si**
20: **Fin Pour**
21: **Retournez** la partition $P^{Ov} = \{P_1^{Ov}, ..., P_{\mathcal{C}}^{Ov}\}$.

varier α dans un intervalle et avec un pas spécifique, nous pouvons obtenir un dendrogramme chevauchant.

3.4.4 Expérimentations sur CDLPOV

Nous testons nos méthodes sur des graphes réels sociologiques que nous avons exposés en introduction. Pour l'évaluation de nos méthodes, nous considérons l'information mutuelle normalisée dans sa forme chevauchante (NMI), le F-Score dans sa forme chevauchante (F_1), l'indice d'omega (Ω) et la modularité de Nicosia (Q_{Ov}^{Nic}). D'autres informations sont données telles que le pourcentage de candidats au chevauchement (cand), le pourcentage de nœuds chevauchants (candOv) et le pourcentage d'arêtes entre communautés (AEC).

Pour analyser les résultats des deux méthodes proposées, nous utilisons des réseaux réels dont certains sont connus comme ayant des nœuds chevauchants. A ce titre, nous exposons les caractéristiques portant sur le chevauchement des réseaux que nous utilisons. Nous mettons également les figures pour certains.

Le graphe de Zachary Karaté club est connu comme ayant deux communautés, et même trois pour certains. La littérature montre l'existence de deux nœuds appartenant à plusieurs communautés. Le graphe de Zachary s'est construit sur une dispute opposant le manager et l'entraîneur d'un club de karaté. Le résultat de cette dispute fut la création de deux clubs de karaté dans une même rue. Le premier nœud qui est chevauchant est ami à la fois avec le manager et l'entraineur du club. Il possède un lien dans chacune des deux communautés. Le second nœud chevauchant est interne à une grande communauté, que l'on peut subdiviser en deux. Il s'agit d'un groupe de personnes amies entre elles dont l'unique connexion au groupe est une personne amie du manager. Ce nœud appartient à un chevauchement.

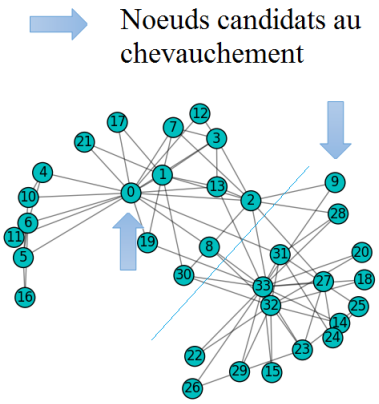


Figure 3.46 – Les nœuds chevauchants dans le graphe de Zachary

Le réseau de dauphins est caractérisé par deux communautés représentant d'un côté les mâles et de l'autre les femelles. C'est un graphe où il n'y a pas de chevauchement.

Le réseau footbalistique représente 11 conférences. On considère qu'il y a huit clubs qui appartiennent à plusieurs communautés.

Le graphe des livres politiques, des années 50 à l'élection présidentielle américaine de 2004 est constitué de trois communautés, à savoir les républicains, les neutres et les démocrates. Les livres politiquement neutres sont une rétrospective de la politique américaine, notamment par des explications historiques. La politique américaine a connu une succession de différentes politiques démocrates et républicaines. Depuis 1961 et jusqu'en 2004, quatre présidents ont été démocrates et cinq présidents ont été républicains. Certains livres du réseau citent des périodes présidentielles marquées par des politiques différentes, menées par des partis différents. C'est par exemple le cas de "Ghost wars", de Steve Coll qui retrace l'histoire de la CIA depuis les cinquantes dernières années. Nous nous attendons à ce que ce livre puisse appartenir aux trois communautés que sont les républicains, les neutres et les démocrates.

Le réseau de collaboration scientifique montre des communautés caractérisées par un thème scientifique. Cependant, certains chercheurs ont publié dans divers domaines et sont susceptibles d'appartenir à plusieurs communautés. Par exemple, Mark Newman, physicien américain, publie à la fois dans le domaine de la détection de communautés mais également dans divers domaines liés au graphe comme les hypergraphes. Cependant, le graphe qui a été formé a déjà été bien défini au sens communautaire et il n'y a que peu de chevauchement.

Zachary Karate Club

Le graphe est caractérisé par deux communautés.

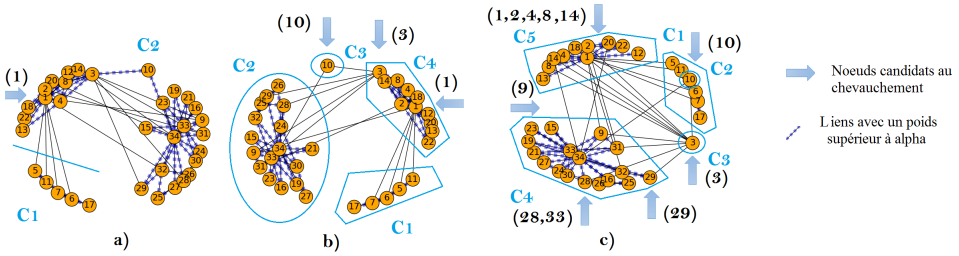


Figure 3.47 – Communautés avec différentes valeurs de α en utilisant f_d et f_{cc}, a) $\alpha \geq 0.6$, b) $\alpha \geq 0.7$ c) $\alpha \geq 0.8$

Sur la figure 3.47, les candidats pour le chevauchement avec f_d et f_{cc} sont

les mêmes. Pour $\alpha \leq 0.6$, seul le nœud 10 est chevauchant. Il est assigné à la communauté C_2 avec f_{cc} alors qu'il est assigné à deux communautés avec f_d (qui sont C_3 et C_4). Cela vient du fait que la communauté C_2 a un nombre plus important de triangles que C_4.

Pour $\alpha \geq 0.8$, le nœud 3 qui est connu dans la littérature comme étant chevauchant, est assigné à deux communautés avec f_d (C_5 et C_4), mais juste à une communauté avec f_{cc} (C_4). Le nœud 1 est assigné à une communauté (C_2) pour chacune des deux méthodes.

D'après les résultats du tableau 3.24, plus la valeur de α est élevée, plus le nombre de candidats pour le chevauchement devient important. Cela vient du fait que les tailles des communautés diminuent alors que α croît. De ce fait, le pourcentage AEC devient plus important, augmentant les nœuds candidats. Même si les nœuds chevauchants sont les mêmes selon f_d et f_{cc} jusque $\alpha \geq 0.9$, la qualité des résultats est meilleure en utilisant f_d plutôt que f_{cc}. La valeur de la modularité la plus élevée est pour $\alpha \geq 0.7$ (0.62 pour chacune des méthodes) avec les valeurs les plus fortes pour le NMI et pour l'indice Ω.

Résultat avec f_d et f_{cc} sur le club de Karate								
f_d	Cand	CandOv	AEC	Q_{Ov}^{Nic}	Ω	F_1	NMI	#
$\alpha \geq 0.6$	47.058%	2.94% (1)	17.95%	0.399	0.065	0.65	0.237	2
$\alpha \geq 0.7$	41.17%	8.824% (3)	16.0%	0.621	0.711	0.857	0.518	4
$\alpha \geq 0.8$	55.88%	32.352% (11)	26.92%	0.420	0.492	0.75	0.349	5
$\alpha \geq 0.9$	55.88%	32.352% (11)	26.92%	0.420	0.492	0.75	0.349	5
f_{cc}	Cand	CandOv	AEC	Q_{Ov}^{Nic}	Ω	F_1	NMI	#
$\alpha \geq 0.6$	47.058%	2.941% (1)	17.948%	0.399	0.064	0.65	0.237	2
$\alpha \geq 0.7$	41.176%	8.823% (3)	16.0%	0.621	0.711	0.857	0.518	3
$\alpha \geq 0.8$	55.882%	32.353% (11)	26.923%	0.420	0.492	0.75	0.349	5
$\alpha \geq 0.9$	55.882%	32.353% (11)	26.923%	0.420	0.492	0.75	0.349	5

Tableau 3.24 – Cand : candidats possibles , CandOv : Pourcentage de nœuds chevauchants

Les femmes du sud de Davis (réseau bipartie)

Il s'agit d'un réseau représentant 18 femmes, observées sur une période de 9 mois dans les années 1930. Durant cette période, de nombreux sous-groupes de ces femmes vont se rencontrer dans une série de 14 évènements sociaux informels. Chacun des nœuds dans ce réseau à deux modes a un nom, représentant une femme ou un évènement (avec la notation E_k pour le $k^{\text{ème}}$ évènement).

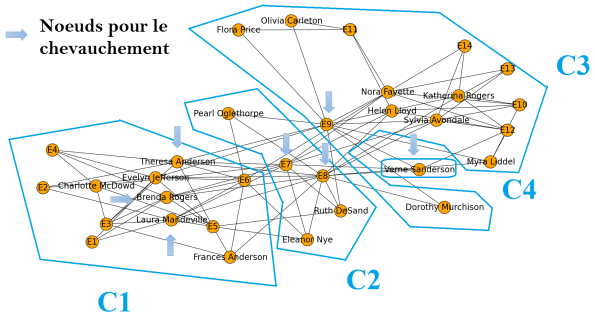


Figure 3.48 – Résultat sur le reseau FM avec $\alpha \geq 0.9$

Résultat avec f_d sur FM					
	Cand	CandOv	AEC	Q_{Ov}^{Nic}	#
$\alpha \geq 0.9$	71.875%	25.0% (5)	33.33%	0.502∗	4
$\alpha \geq 1.0$	87.5%	75.0% (14)	56.99%	0.021∗	17
Résultat avec f_{cc} sur FM					
	Cand	CandOv	AEC	Q_{Ov}^{Nic}	#
$\alpha \geq 0.9$	71.875%	65.625% (12)	33.33%	0.157*	4
$\alpha \geq 1.0$	87.5%	87.5% (16)	56.989%	0.0∗	17

Tableau 3.25 – Cand : candidats possibles, CandOv : Pourcentage de nœuds chevauchants

En considérant le Tableau 3.25, pour $\alpha \geq 0,8$, nous n'avons qu'une communauté. Pour $\alpha \geq 0,9$, sur la figure 3.48, nous obtenons quatre communautés. Après avoir calculé f_d, nous avons 25% de nœuds dans des communautés chevauchantes. Comme nous pouvons le voir, c'est à la fois les nœuds représentant des individus et des évènements qui sont assignés à des communautés chevauchantes. Verne est assigné à $E7$, $E8$, $E9$ et $E12$. $E7$ et $E9$ sont assignés à Theresa, Laura et Brenda (dans la communauté C_1), puis pour Verne dans C_4 et finalement pour Hélène et Sylvia C_3, soit à trois communautés différentes. Theresa est assignée aux activités $E6$, $E7$ et $E8$. A la fin du processus, certains nœuds sont liés à plusieurs évènements comme Verne avec $E7$, $E8$, $E9$ et $E12$. pour $\alpha = 1,0$, nous obtenons 17 communautés. La majorité des communautés regroupe des personnes avec leurs évènements qui sont assignés aux personnes ayant de fortes centralités.

On observe qu'utiliser f_{cc} entraîne un pourcentage de chevauchement plus important qu'en utilisant f_d (65.625% contre 25.0%), avec un taux de chevauchement moyen plus important, particulièrement pour les évènements.

Les dauphins de Nouvelle Zélande

Le graphe est composé de deux communautés, qui regroupent les mâles et les femelles.

Résultats avec f_d et f_{cc} sur le réseau de dauphins								
f_d	Cand	CandOv	AEC	Q_{Ov}^{Nic}	Ω	F_1	NMI	#
$\alpha \geq 0.5$	51.61% %	0.0%	20.38%	0.7959	1.0	1.0	1.0	2
$\alpha \geq 0.6$	54.838%	6.451% (4)	24.050%	0.750	0.617	.857	0.594	4
$\alpha \geq 0.7$	64.51%	8.065% (5)	30.57%	0.714	0.478	0.75	0.457	5
$\alpha \geq 0.8$	61.29%	19.355% (12)	29.30%	0.605	0.478	0.6184	0.442	8
$\alpha \geq 0.9$	77.41%	25.81% (16)	43.94%	0.542	0.355	0.533	0.246	12
f_{cc}	Cand	CandOv	AEC	Q_{Ov}^{Nic}	Ω	F_1	NMI	#
$\alpha \geq 0.5$	51.613%	0.0%	20.382%	0.796	1.0	1.0	1.0	2
$\alpha \geq 0.6$	54.838%	6.451% (4)	24.051%	0.750	0.613	.857	0.594	4
$\alpha \geq 0.7$	64.516%	8.065% (5)	30.57%	0.714	0.429	0.75	0.457	5
$\alpha \geq 0.8$	61.29%	19.355% (12)	29.3%	0.606	0.478	0.618	0.442	8
$\alpha \geq 0.9$	77.419%	35.483% (22)	43.949%	0.441	0.588	0.549	0.277	12

Tableau 3.26 – Cand : candidats possibles, CandOv : Pourcentage de nœuds chevauchants

A partir du tableau 3.26, pour $\alpha \geq 0.5$, l'algorithme et les fonctions trouvent les deux communautés, sans aucun chevauchement, avec un NMI, un indice d'omega et un F_1 score de 1,0. En augmentant la valeur de α, la taille des communautés diminue tandis que le nombre de candidats possibles pour le chevauchement augmente. Le pourcentage de chevauchement est le même pour $\alpha \geq 0.6$ jusqu'à $\alpha \geq 0.8$. Malgré cela, les deux méthodes n'assignent pas les nœuds candidats de la même manière. f_{cc} produit la même qualité en termes de communautés mais a un pourcentage de chevauchement et un taux de chevauchement moyen plus important qu'avec f_d, surtout pour de fortes valeurs de α.

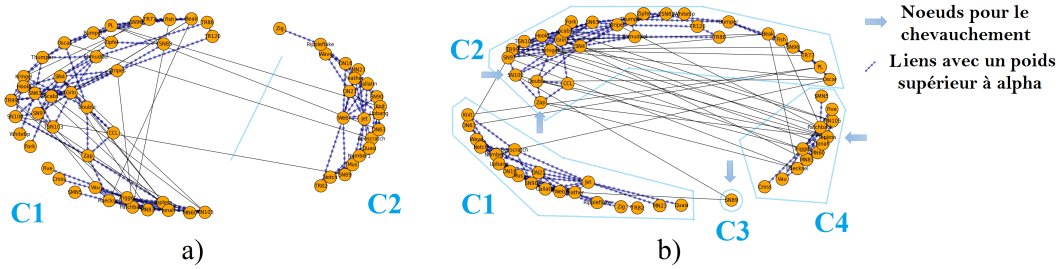


Figure 3.49 – Réseaux de dauphins a) $\alpha \geq 0,5$, b) $\alpha \geq 0,6$, les nœuds chevauchants avec f_{cc} et f_{dd} sont les mêmes

En observant la figure 3.49, pour $\alpha \geq 0.6$, nous voyons que 4 nœuds ont été selectionnés pour appartenir à plus d'une communauté. En considérant leurs aspects topologiques, $SN89$ est lié à deux communautés avec une arête liée à chacune, il est assigné à ses communautés voisines. Il en est de même pour $Topless$ qui a 6 arêtes à l'intérieur de sa communauté et 4 à l'extérieur. Zap a trois nœuds à l'intérieur de sa communauté et 2 à l'extérieur. Il est assigné à C_4. $SN100$ a 4 arêtes à l'intérieur de sa communauté et deux à l'extérieur vers deux communautés. Il est assigné aux communautés C_3 et C_4. $SN9$ a juste une arête

de plus que $SN100$ à l'intérieur de la communauté avec la même configuration topologique, il n'est cependant pas assigné. Pour $\alpha \geq 0.6$, les résultats, que cela soit en utilisant f_d ou f_{cc} sont les mêmes. Pour chacune des deux méthodes, la taille des dendrogrammes est de 6. De $\alpha \geq 0.5$ à $\alpha \geq 0.9$, le nombre de nœuds sélectionnés pour le chevauchement est le même, mais le pourcentage de chevauchement avec f_{cc} est plus important avec f_d.

Le réseau footballistique

Résultats avec f_d sur le réseau footballistique								
f_d	Cand	CandOv	EC	Q_{Ov}^{Nic}	Ω	F_1	NMI	#
$\alpha \geq 0.2$	100.0%	0.0%	29.53%	0.722	0.530	0.762	0.597	9
$\alpha \geq 0.3$	100.0%	0.0%	29.53%	0.708	0.681	0.810	.639	10
$\alpha \geq 0.4$	100.0%	0.0%	30.01%	0.7	0 .865	0.854	0.685	11
$\alpha \geq 0.5$	100.0%	0.87% (1)	30.01%	0.699	0.851	0.854	0.682	11
$\alpha \geq 0.6$	100.0%	1.74% (2)	30.83%	0.690	0.882	0.861	0.666	12
$\alpha \geq 0.7$	100.0%	8.69% (10)	31.32%	0.629	0.825	0.819	0.629	13
$\alpha \geq 0.8$	100.0%	8.69% (10)	31.32%	0.629	0.825	0.819	0.629	13
f_{cc}	Cand	CandOv	EC	Q_{Ov}^{Nic}	Ω	F_1	NMI	#
$\alpha \geq 0.2$	100.0%	0.0%	29.53%	0.722	0.530	0.76	0.512	9
$\alpha \geq 0.3$	100.0%	0.0%	29.53%	0.708	0.681	0.81	0.639	10
$\alpha \geq 0.4$	100.0%	0.0%	30.016%	0.699	0.865	0.854	0.685	11
$\alpha \geq 0.5$	100.0%	0.87% (1)	30.016%	0.699	0.851	0.854	0.682	11
$\alpha \geq 0.6$	100.0%	1.74% (2)	30.832%	0.690	0.882	0.85	0.666	12
$\alpha \geq 0.7$	100.0%	8.69% (10)	31.321%	0.629	0.825	0.819	0.629	13
$\alpha \geq 0.8$	100.0%	8.69% (10)	31.32%	0.629	0.825	0.819	0.629	13

Tableau 3.27 – Cand : Candidats possibles, CandOv : Pourcentage de nœuds chevauchants

Selon le tableau 3.27, c'est pour $\alpha \geq 0,5$ que les premiers nœuds chevauchants apparaissent. De $\alpha \geq 0,5$ à $\alpha \geq 0.9$, le nombre de nœuds chevauchants est le même, ainsi que leur taux de chevauchement moyen. Les résultats en termes de similarité de score de qualités sont semblables. les meilleurs scores sont pour $\alpha \geq 0,5$ avec un NMI 0.68. Le nombre de communautés augmente très lentement puis fortement avec $\alpha = 1.0$.

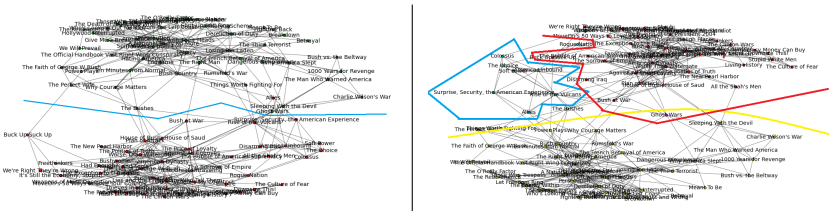
Les livres politiques de Kreb

C'est un réseau de livres politiques datant de l'élection présidentielle américaine de 2004 et vendus sur le site de vente en ligne *Amazon.com*. Ce graphe comporte trois communautés au sens politique, à savoir les démocrates, les républicains et le centre sur l'échiquier politique.

Resultats avec f_{cc} et f_d sur les livres politiques de Kreb								
f_d	Cand	CandOv	EBC	Q_{Ov}^{Nic}	Ω	F_1	NMI	#
$\alpha \geq 0.4$	24.762%	0.0%	5.215%	0.834	0.667	0.788	0.452	2
$\alpha \geq 0.5$	26.67%	0.95% (1)	6.576%	0.834	0.654	0.784	0.494	2
$\alpha \geq 0.6$	31.43%	1.90% (2)	7.71%	0.845	0.676	0.713	0.387	3
$\alpha \geq 0.7$	32.38%	5.71% (6)	9.07%	0.76	0.686	0.664	0.354	4
$\alpha \geq 0.8$	35.24%	15.24% (16)	9.98%	0.653	0.667	0.566	0.290	7
f_d	Cand	CandOv	EBC	Q_{Ov}^{Nic}	Ω	F_1	NMI	#
$\alpha \geq 0.4$	24.761%	0.0%	5.215%	0.834	0.667	0.788	0.504	2
$\alpha \geq 0.5$	26.67%	0.952% (1)	6.576%	0.834	0.654	0.774	0.494	2
$\alpha \geq 0.6$	31.428%	0.952% (1)	7.709%	0.844	0.676	0.719	0.449	3
$\alpha \geq 0.7$	32.380%	5.714% (6)	9.070%	0.782	0.687	0.653	0.340	4
$\alpha \geq 0.8$	35.238%	15.238% (16)	9.977%	0.6533	0.667	0.532	0.28	7

Tableau 3.28 – Cand : Candidats possibles , CandOv : Pourcentage de nœuds chevauchants

D'après le tableau 3.30, c'est pour $\alpha \geq 0.5$ que les premiers nœuds chevauchants apparaissent. Les résultats sont très similaires pour les deux méthodes, néanmoins, f_d donne un pourcentage de chevauchement et un taux de chevauchement moyen plus élevé selon nos observations qu'en utilisant f_{cc}. Le nombre de communautés augmente lentement en fonction de la valeur α, comme celui des candidats au chevauchement. De $0.4 \leq \alpha \leq 0.8$, la majorité des nœuds chevauchants sont neutres sur le plan politique américain.



(a) Le réseau de livres politiques avec $\alpha \geq 0,3$

(b) Le réseau de livres politiques avec $\alpha \geq 0,6$

Figure 3.50 – Résultats avec f_d et f_{cc}

Sur la figure 3.50a, nous voyons que la méthode pour $\alpha \geq 0.3$ donne deux communautés qui représentent les républicains et les démocrates, il n'y a pas de chevauchements. Sur la figure 3.50b, avec $\alpha \geq 0.6$, la méthode trouve trois

communautés, les républicains, les démocrates et les neutres et un nœud chevauchant, mais qui n'est pas associé aux communautés de la même façon selon les fonctions d'appartenance utilisées. Il s'agit du livre "Ghost Wars" écrit par Steve Coll et publié en 2004. Ce livre retrace l'action de la CIA des années 1965 à 2004, sous les différentes administrations américaines, contre des réseaux criminels. Ce livre n'a pas d'orientation politique et n'est qu'historique. Les tableaux 3.29 et 3.30 montrent certains livres chevauchant plusieurs communautés. Pour Ghost Wars, le livre est neutre pour $\alpha \geq 0.5$, puis chevauche la communauté républicaine avec $\alpha \geq 0.6$ et finalement chevauche également la communauté démocrate avec $\alpha \geq 0.7$, quelles que soient les fonctions utilisées. Ce livre est chevauchant car il est cité de nombreuses fois par les communautés républicaines et démocrates. En effet, des années 1965 à 2004, les différentes administrations américaines étaient soit démocrates (Lyndon B. Johnson, Jimmy Carter et Bill Clinton), soit républicaines (Richard M. Nixon, Gerald R. Ford, Ronald W. Reagan, George H. W. Bush). Néanmoins,il existe plus de liens vers la communauté républicaine (4) que vers la communauté démocrate (3), c'est l'une des raisons qui ont pu le placer dans le parti républicain lors du premier chevauchement. Les autres livres ont une appartenance politique, mais restent proches de la frontière des communautés. Ils sont chevauchés avec la communauté neutre.

Resultats avec f_d				
Livres	Démocrates	Neutres	Républicains	α
Ghost Wars		√		0.5
Ghost Wars		√	√	0.6
Ghost Wars	√	√	√	0.7
Disarming	√		√	0.6
Disarming	√	√	√	0.7
Rise of the Vulcans		√	√	0.7
The choice	√	√	√	0.7
The great unraveling	√	√		0.7
American Dynasty	√	√		0.7

Tableau 3.29 – Les livres chevauchants

Resultats avec f_{cc}				
Livres	Démocrates	Neutres	Républicains	α
Ghost Wars		√		0.5
Ghost Wars		√	√	0.6
Ghost Wars	√	√	√	0.7
Disarming		√		0.7
Disarming	√	√		0.7
Rise of the Vulcans		√	√	0.7
The choice	√	√	√	0.7
The great unraveling	√	√		0.7
American Dynasty	√	√		0.7

Tableau 3.30 – Les livres chevauchants

Les réseaux de collaborations scientifiques de Newman (Netscience)

Ce graphe représentant des collaborations scientifiques entre chercheurs est caractérisé par une très faible densité 0.0021. Les tailles de communautés détectées sont très petites. Observant le tableau 3.31, c'est pour $\alpha \geq 0.4$ que les premiers nœuds chevauchants apparaissent. Le nombre d'arêtes entre communautés est relativement faible, de 2.39% d'arêtes avec $\alpha \geq 0.2$ à 18.27% avec $\alpha \geq 0.8$, impliquant un faible pourcentage de candidats. Seulement 0.61% de nœuds sont candidats avec $\alpha \geq 0.4$ tandis que l'on obtient 6.36% pour f_d. Pour chacune des méthodes, le pourcentage de chevauchement est très similaire et relativement faible. Observant la modularité, les résultats sont très similaires. On peut conclure que pour des graphes faiblement denses, les résultats sont très similaires en utilisant les fonctions fondées sur la densité ou sur le coefficient de clustering.

Resultats avec f_d sur Netscience					
f_d	Cand	CandOv (nombre)	AEC	Q_{Ov}^{Nic}	#
$\alpha \geq 0.2$	3.285%	0.0%	2.396%	0.977	293
$\alpha \geq 0.3$	5.544%	0.0%	4.175%	0.972	297
$\alpha \geq 0.4$	7.392%	0.616% (9)	5.79%	0.948	308
$\alpha \geq 0.5$	9.240%	0.684% (10)	7.734%	0.940	315
$\alpha \geq 0.6$	14.099%	2.396% (35)	12.731%	0.886	342
$\alpha \geq 0.7$	16.016%	4.517% (72)	15.332%	0.845	360
$\alpha \geq 0.8$	18.07%	6.366% (101)	18.275%	0.817	371
f_{cc}	Cand	CandOv (nombre)	AEC	Q_{Ov}^{Nic}	#
$\alpha \geq 0.2$	3.285%	0.0%	2.396%	0.977	293
$\alpha \geq 0.3$	5.544%	0.0%	4.175%	0.972	297
$\alpha \geq 0.4$	7.392%	0.548% (8)	3.146%	0.949	308
$\alpha \geq 0.5$	9.240%	0.616% (9)	4.232%	0.942	315
$\alpha \geq 0.6$	14.1%	2.164% (36)	6.966%	0.886	342
$\alpha \geq 0.7$	16.016%	5.339% (85)	8.39%	0.855	360
$\alpha \geq 0.8$	18.07%	7.05% (112)	10.0%	0.813	371

Tableau 3.31 – Cand : Candidats possibles, CandOv : Pourcentage de nœuds chevauchants

3.4.5 Etude portant sur le temps d'exécution

Le temps d'exécution de l'algorithme dépend à la fois du temps nécessaire au calcul des $\mathcal{N}$ propagations de labels alimentant la matrice de fréquence Δ_{CDLP} (incluant le temps de création du nouveau graphe qui induit la recherche des composantes connexes qui sont nos communautés) et le temps de calcul de la fonction d'appartenance $\Delta_{fonction}$. Le temps de lecture et d'écriture des fichiers

n'est pas pris en compte. Le temps d'exécution est ainsi décomposé de la manière suivante : $\Delta_T = \Delta_{CDLP} + \Delta_{fonction}$.

Concernant le temps d'exécution du CDLP, figure *a*) 3.51, le temps est presque constant, avec une très légère diminution lorsque α augmente. Plus α est élevé, plus le nombre d'arêtes du graphe seuillé devient faible. Par conséquence, le temps d'exécution pour le calcul des composantes connexes diminue.

Pour la fonction d'appartenance fondée sur la densité des communautés, figure *b*) 3.51, en observant tous les réseaux, le temps est quasi-linéaire jusqu'à ce qu'une cassure intervienne à $\alpha \leq 0.8$, où parallèlement le temps augmente soudainement, de même que l'AEC et par voie de conséquence, le nombre de candidats potentiels au chevauchement. Le nombre de combinaisons devenant de plus en plus important, le temps augmente. Nous voyons également que plus la densité du graphe est élevée, plus le temps de calcul de f_d est important.

Concernant la fonction d'appartenance fondée sur le cœfficient de clustering, deux choses sont à considérer : le temps de calcul du CCL et le nombre potentiel de candidats au chevauchement. D'après la figure *b*) 3.51, les temps ne sont pas constants. Pour le réseau des dauphins, le temps augmente jusqu'à $\alpha \geq 0.5$, et diminue juste après. Cela vient du fait que dans les grosses communautés, le temps de calcul du CCL est plus important. Avec une forte valeur de α, le temps de calcul du CCL diminue alors que le nombre de candidats potentiels au chevauchement augmente, avec un nombre de combinaisons plus important pour la fonction d'appartenance. Pour $\alpha \geq 0.8$, le temps d'exécution augmente brutalement.

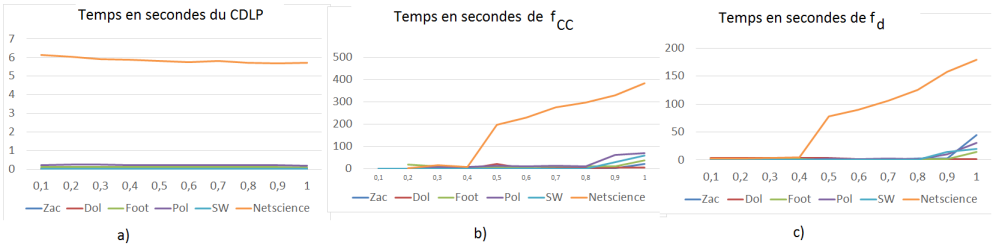


Figure 3.51 – Temps d'exécution du CDLP, l'axe horizontal représentant la valeur α et l'axe vertical représentant le temps en secondes. a) Temps du CDLP b) Temps de f_d c) Temps du f_{cc}

3.4.6 Analyse comparative

Nous comparons nos propositions algorithmiques avec celles issues de la littérature les plus utilisées, à savoir : CFinder Palla *et al.* (2005), COPRA ($\nu = 2$ et $\nu = 3$ Gregory (2010), ν étant le nombre de communautés auquel un nœud appartient), OSLOM Lancichinetti *et al.* (2011), SLPA Xie *et al.* (2011),

et CONGA Gregory (2007).

Analyse comparative						
Réseaux	F_1	Ω	NMI	Q_{Ov}^{Nic}	#	%
Zac #2						
CFinder	0.48	0.35	0.18	0.52	3	5.88%
OSLOM	**0.86**	**0.84**	**0.80**	**0.748**	2	2.94%
CONGA	0.65	0.113	0.274	0.441	2	2.94%
$COPRA_2$*	0.281	0.266	0.228	0.414	11.3	5.58%
$COPRA_3$*	0.684	0.359	0.347	0.452	6.4	12.64%
SLPA*	0.86	0.633	0.564	**0.608**	2.12	2.20%
CDLPOV f_d*	0.852	0.711	**0.518**	0.621	4	8.82%
CDLPOV f_{cc}*	0.852	0.711	**0.518**	0.621	4	8.82%
Foot #12						
CFinder	0.701	0.64	0.55	0.51	13	6.9%
OSLOM	**0.954**	**0.802**	**0.759**	0.696	12	0.0%
CONGA	0.823	0.321	0.423	0.451	11	60.0%
$COPRA_2$*	0.933	0.788	0.705	0.693	10.8	0.52%
$COPRA_3$*	0.944	0.747	0.712	0.668	11.2	2.52%
SLPA*	0.748	0.684	0.612	**0.715**	10.30	1.69%
CDLPOV f_d*	0.854	0.865	**0.751**	0.699	11	0.0%
CDLPOV f_{cc}*	0.854	0.865	**0.751**	0.699	11	0.0%
SW						
CDLPOV f_d*				0.502∗	4	25.0%
CDLPOV f_{cc}*				0.157∗	4	65.6%
#2						
CFinder	0.57	0.35	0.26	0.66	4	3.72%
OSLOM	**1.0**	**0.914**	**0.852**	0.742	2	1.61%
CONGA	0.85	0.892	0.821	0.746	2	3.22%
$COPRA_2$*	0.933	0.788	0.751	0.693	10.8	0.52%
$COPRA_3$*	0.893	0.767	0.701	0.677	3.7	7.73%
SLPA*	0.56	0.754	0.632	0.742	3.44	2.00%
CDLPOV f_d*	**1.0**	**1.0**	**1.0**	**0.796**	2	0.0%
CDLPOV f_{cc}*	**1.0**	**1.0**	**1.0**	**0.796**	2	0.0%
Pol #4						
CFinder	**0.855**	**0.740**	**0.79**	0.884	4	(9)
OSLOM	**0.814**	**0.704**	**0.55**	**0.847**	2	1.90%
CONGA	0.688	0.651	0.49	0.779	4	4.16%
$COPRA_2$*	0.687	0.637	0.385	0.825	3	1.05%
$COPRA_3$*	0.702	0.649	0.416	0.827	2.8	6.47%
SLPA*	0.755	0.648	0.497	0.83	3.40	12.5%
CDLPOV f_d*	0.784	0.654	0.495	0.844	3	1.90%
CDLPOV f_{cc}*	**0.788**	**0.667**	**0.503**	**0.834**	2	0.0%
NS						
CDLPOV f_d*				0.977	293	0.0%
CDLPOV f_{cc}*				0.977	293	0.0%

Tableau 3.32 – (*) algorithmes à base de propagation de labels

Nous montrons les résultats de nos méthodes donnant les scores des mesures non supervisées les plus élévés dans le tableau 3.32. Nos propositions algorithmiques donnent d'assez bons résultats en termes de qualité. Nous obtenons de meilleurs résultats que COPRA et une meilleure stabilisation. Même si les algorithmes à base de propagation de labels produisent en moyenne plus de communautés, le CDLP avec f_d et f_{cc} en produit moins. Nous expliquons ce fait par la présence de la matrice de fréquence qui stabilise la propagation de labels.

3.4.7 Conclusion sur les propositions algorithmiques chevauchantes

Nous avons proposé deux méthodes appliquées à la propagation de label par cœurs pour la détection de communautés chevauchantes. Chacune de ces méthodes utilise la matrice de fréquence et les caractéristiques sociales et topologiques des communautés pour savoir si certains nœuds peuvent appartenir à plusieurs communautés. Les utilisateurs ont le choix entre laisser l'algorithme assigner de possibles candidats au chevauchement à un nombre spécifique de communautés auxquelles ces nœuds appartiendraient, ou laisser les fonctions d'appartenance tester toutes les combinaisons pour des assignations automatiques. Les fonctions d'appartenances sont fondées sur la densité (f_d), le CC (f_{cc}) et sur les communautés détectées. Les résultats selon les différentes fonctions sont assez similaires en termes de qualité. Néanmoins, le taux d'assignation de nœuds chevauchants à un plus grand nombre de communautés est plus important avec f_d qu'avec f_{cc}. Concernant le temps d'exécution, plus la densité du graphe étudié est importante, plus le nombre de candidats est grand, augmentant ainsi le nombre de combinaisons à tester pour nos fonctions, et par conséquence le temps augmentera. Nous avons vu que calculer les 100 LPA pour alimenter la matrice de fréquence est assez rapide, (1 seconde pour Zachary et 6 secondes pour NS). Néanmois, le temps de calcul de f_{cc} est plus important que celui de f_d. Cela vient du calcul du nombre de triangles au sein des communautés pour calculer le CCL. Le temps de calculs des deux fonctions augmente parallèlement à α. Mais à partir d'un seuil ($\alpha \geq 0.8$ ou $\alpha \geq 0.9$ selon les graphes testés), le temps devient très important. Pour $\alpha \geq 0.5$, nous avons besoin de 30 secondes pour calculer f_d et 60 secondes sur NS. Pour $\alpha \geq 0.8$, nous avons besoin d'une centaine de secondes pour calculer f_d et f_{cc} toujours sur NS. Pour les réseaux portant sur Zachary, les dauphins, SW ou les livres politiques, il faut approximativement 10 secondes pour f_d et près de 20 secondes avec f_{cc}. Les perspectives seront présentées au chapitre 4.

Chapitre 4

Propositions algorithmiques sur la détection de communautés parallèles et distribuées

Sommaire

Dans le chapitre précédent, nous avions présenté des propositions algorithmiques permettant la détection de communautés disjointes et chevauchantes. Cependant, dans un monde où le volume des données devient de plus en plus important, il est utile de pouvoir retranscrire nos propositions algorithmiques pour de grands graphes.

Ce chapitre aura comme objectif de présenter une version parallèle et distribuée du CDLP sous Apache Hadoop. Nous testerons notre framework sur

quatre grands graphes de la littérature et effectuerons une étude comparative avec d'autres algorithmes issus de la littérature.

4.1 Propositions sur le parallélisme et la distribution

Apache Hadoop est une architecture permettant d'effectuer du calcul parallèle et distribué pour répondre aux traitements des données de taille massive. Comme nous l'avons vu au chapitre 1, Hadoop possède un framework MapReduce composé de deux fonctions, un *mapper* et un *reducer*.

La propagation de labels peut s'effectuer de manière synchrone ou asynchrone (cette dernière étant plus stable et offrant une meilleure qualité de partitionnement). Cependant, la méthode asynchrone nécessite la mise en place d'un ordre de visite sur les nœuds (qui peut être aléatoire) afin d'effectuer les changements de labels pour chacun des nœuds. Cette opération s'effectue d'ordinaire de manière séquentielle. C'est-à-dire que lorsqu'un nœud a son label modifié, les autres nœuds du graphe sont informés et un autre nœud change son label par un vote majoritaire en utilisant la nouvelle information. Cela peut être fait pour de petits graphes, mais pas pour de grands réseaux. Il ne serait pas souhaitable d'avoir un graphe de plusieurs centaines de milliers de nœuds et de visiter séquentiellement chacun d'entre eux pour en modifier le label. C'est en ce sens que nous proposons d'utiliser la méthode semi-synchrone pour effectuer un changement de label qui puisse être parallèle et distribué. Pour rendre l'algorithme semi-synchrone plus déterministe, nous utilisons la matrice de cofréquence. La propagation de labels semi-synchrone consiste à colorier le graphe de telle sorte que chaque nœud du graphe ait une couleur différente de celle de ses voisins. La propagation de labels se fait en suivant les nœuds ayant une même couleur. Ainsi, lorsqu'un nœud met à jour son label par un vote majoritaire, son label est changé alors que celui de ses nœuds voisins ne l'est pas. Ce processus se fait sur tous les nœuds ayant la même couleur. Puis, on fait savoir aux nœuds voisins le nouveau label du nœud dernièrement modifié. On passe alors à une autre couleur pour continuer la propagation de label. Le processus consiste donc à colorier, propager les labels des nœuds ayant une couleur donnée, faire une mise à jour des voisins des nœuds ayant subi une modification de leurs labels puis répéter les phases de propagation de labels et de mise à jour sur les nœuds restants. Ce sont ces opérations que nous avons mises sous forme MapReduce pour paralléliser et distribuer le CDLP. Nous rappelons que le CDLP a deux paramètres principaux qui sont le seuil α (utilisé après l'alimentation de la matrice de fréquence pour créer un nouveau graphe pondéré) et le nombre de propagations de labels ($\mathcal{N}$) lancées en parallèle. En utilisant le seuil α, les composantes connexes ainsi créées représentent les communautés.

4.1.1 Proposition algorithmique pour la propagation de labels parallèle et distribuée avec détection de cœurs

Nous exposons la propagation de labels parallèle et distribuée avec détection de cœurs, Algorithme 8. Il s'agit de la version parallèle et distribuée du CDLP présenté au Chapitre 2. La complexité de l'algorithme est en $\mathcal{O}(\mathcal{N} \times k \times (m) + n + m + n^2 \times m)$.

Algorithme 8 La propagation de labels semi-synchrone parallèle et distribuée avec détection de cœurs PAR-CDLP

Input : Un graphe $G = (V, E)$, un seuil α, $\mathcal{N}$ le nombre de propagations de labels

Output : Les communautés trouvées par l'algorithme sur le graphe $G = (V, E)$

1: Effectuer une coloration du graphe G, $\mathcal{D} = \{D_1, D_2, ..., D_l\}$ (de l parties)
2: Allouer une matrice de co-fréquence vide $P_{ij}^{\mathcal{N}}$
3: Appliquer $\mathcal{N}$ fois la propagation de labels semi-synchrone et remplir la matrice $P_{ij}^{\mathcal{N}}$
4: Créer un nouveau graphe $G' = (V, E')$ issu de$P_{ij}^{\mathcal{N}}$, dont les arêtes ont un poids supérieur ou égal à α
5: Créer une partition P en considérant les $\mathcal{C}$ composantes connexes comme cœurs
6: **Retourner** la partition $P = \{P_1, ..., P_{\mathcal{C}}\}$.

4.1.2 Modélisation MapReduce de PAR-CDLP

Notre proposition algorithmique comprend 4 grandes parties :
— la mise en place d'un algorithme de coloriage de graphe
— le lancement de $\mathcal{N}$ propagations de labels semi-synchrones
— l'alimentation d'une matrice de co-fréquence
— la détection des composantes connexes avec un seuil α

Dans la suite de cette section, nous proposons de suivre un petit exemple visuel pour une meilleure compréhension du développement de PAR-CDLP. Nous considérons un fichier d'arêtes où chaque ligne représente un lien $i - j$ (et $j - i$), Figure 4.1.

Etape de coloration de graphe :

Cette étape consiste à attribuer à chaque nœud une couleur de telle sorte que deux nœuds adjacents n'aient pas la même couleur. C'est à partir de cette couleur que la propagation de labels semi-synchrone pourra être effectuée, où les nœuds ayant une même couleur vont changer leur label en fonction de leur voisinage, qui a une couleur différente. Nous commençons avec un fichier contenant

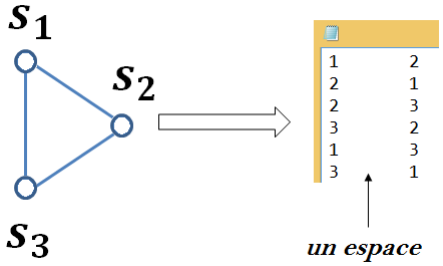


Figure 4.1 – Exemple avec un triangle pour l'illustration de notre proposition algorithmique PAR-CDLP.

toutes les arêtes du graphe sous la forme $i-j$ et $j-i$ comme le montre l'exemple Figure 4.1. Un algorithme de coloration est appliqué. Une implantation d'un tel algorithme a été proposée par Gandhi et Misra (2015). Le résultat est un fichier contenant les résultats de la nouvelle coloration avec les lignes $i-j-C_j$ et $j-i-C_i$, où C_i est la couleur du nœud i, Figure 4.2.

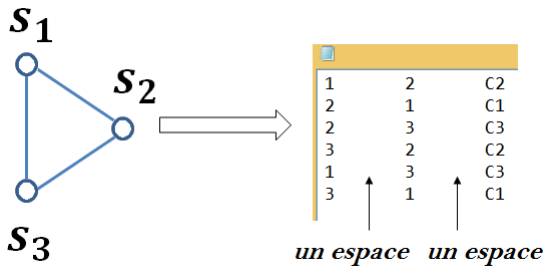


Figure 4.2 – Exemple avec un triangle ayant subi une coloration de graphe

Propagation de label semi-synchrone :
A chaque nœud est associé une vecteur de labels qui permet d'effectuer les $\mathcal{N}$ propagations de labels en parallèle. Chaque couleur servira d'ordre de visite sur le graphe avec l'intention de faire la mise à jour des labels afin de faire la propagation de labels semi-synchrone. Une étape suivante de mise à jour consistera à transmettre les nouveaux labels aux nœuds voisins qui n'ont pas encore connu de mise à jour de leur label.

Le premier job consiste en une fonction `FirstReadMap`, Algorithme 9, et une fonction `FirstReadReduce`, Algorithme 10. Le mapper `FirstReadMap` lit le fichier d'entrée et émet $j-l_i-i-C_i$ où l_i est un vecteur de taille $\mathcal{N}$, contenant le label courant du nœud i pour les $\mathcal{N}$ différentes propagations de labels. Nous notons par $|l_i|$ la taille du vecteur l_i. Initialement, chaque nœud a son propre label pour les $\mathcal{N}$ propagations de labels, $l_i = [i, i, ..., i]$. La ligne

$j - l_i - i - C_i$ signifie que le nœud j va recevoir le vecteur l_i du nœud i. Le nœud j connaîtra donc les labels de son voisin i pour les $\mathcal{N}$ propagations de labels. La phase de shuffling groupe les nœuds selon le nœud j qui reçoit l'information pour la phase de réduction. Les reducers, prenant comme clé spécifique j et comme valeur, les lignes j où était le receveur lors de la phase de map, émettent $j - l_i - C_i$ sans faire la moindre opération, Figure 4.3.

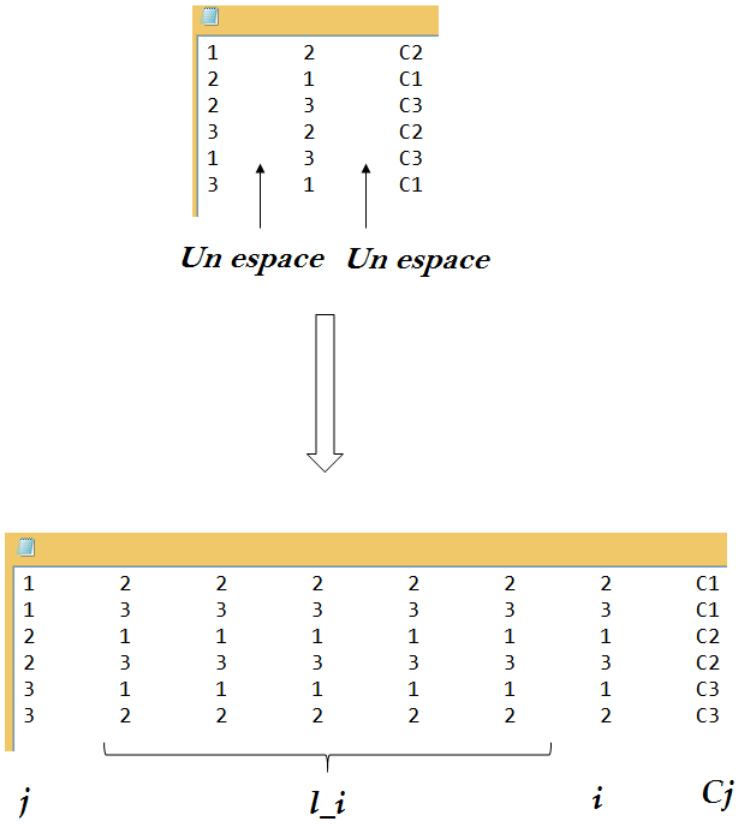


Figure 4.3 – Résultats de la fonction `FirstReadMap` avec $\mathcal{N} = 5$

La seconde partie consiste en une boucle sur deux jobs. Le premier job réalise un vote sur les nœuds de la couleur courante. Cela consiste en `VoteMap`, Algorithme 11 et `VoteReduce`, Algorithme 12. Le second job effectue une mise à jour sur les nœuds ayant transmis l'information ou sur les nœuds concernés par la prochaine propagation de labels (comme les voisins des nœuds ayant subi une modification de leur label). Cela consiste en `UpdateMap`, Algorithme 11 et `UpdateReduce`, Algorithme 13. La boucle sur ces deux jobs se fonde sur la coloration comme compteur (le vote ayant lieu pour les nœuds de la couleur courante).

Algorithme 9 `FirstReadMap`

Entrée : $\mathcal{N}$ le nombre de lancements, Map(Clé : Nom du fichier, Valeur : fichier d'arêtes représentant un graphe $G = (V, E)$ avec la coloration $(i - j - C_j)$)

Sortie : émettre les lignes $(j - l_j - i - C_j)$,avec $|l_j| = \mathcal{N}$

1: $l_j \longleftarrow []$ {vecteur pour les $\mathcal{N}$ propagations de labels différentes}
2: **Pour** ligne dans valeur **Faire**
3: **Pour** $j = 0$ à $\mathcal{N}$ **Faire**
4: $l_j.ajouter(label(i))$ {label(i) est le label du nœud i}
5: **Fin Pour**
6: **Fin Pour**
7: **Emettre**$(j - l_j - i - C_j)$ {la ligne contient $\mathcal{N} + 3$ éléments du fait que l_j contient $\mathcal{N}$ éléments, j, i et C_j, juste un pour chaque}

Algorithme 10 `FirstReadReduce`

Entrée : $\mathcal{N}$ le nombre de lancements, Reduce(Clé : nœud i, Itvaleur : Itérateur [$[l_j - C_j]$,...)

Sortie : émettre les lignes $(i - l_j - j - C_j)$ avec $|l_j| = \mathcal{N}$

1: **Pour** ligne dans Itvaleur **Faire**
2: **Emettre** $(j - line)$ {équivalent à $j - l_j - i - C_j$, avec une longueur de $\mathcal{N} + 3$ éléments, du fait que l_i contient $\mathcal{N}$ éléments, j, i et C_j, une pour chacune}
3: **Fin Pour**

Algorithme 11 `VoteMap,UpdateMap,CommunityMap,GPrimAlphaMap`

Entrée : $\mathcal{N}$ le nombre de lancements, Map(Clé : nom de fichier , Valeur : fichier d'arêtes avec la propagation de labels avec couleurs $(j - l_i - i - C_j)$)

Sortie : émettre des lignes $j - l_j - i - C_j$ (avec $|l_j| = \mathcal{N}$)

1: $l_j \longleftarrow []$ {vecteur pour les $\mathcal{N}$ propagations de labels}
2: **Pour** ligne dans Valeur **Faire**
3: **Emettre**$(j - l_i - i - C_j)$ {ligne contenant $\mathcal{N} + 3$ éléments, du fait que l_i contient $\mathcal{N}$ éléments, j, i et C_j, une pour chaque ligne}
4: **Fin Pour**

Algorithme 12 `VoteReduce`

Entré : $\mathcal{N}$ le nombre de lancements, Reduce(Clé : j, Itérateur *valeurs* : lignes avec la propagation de labels et coloration [[$(l_i - i - C_j)$],...]) {*valeurs* est un vecteur de vecteurs, on note par *valeurs*[k] le k^{th} élément du vecteur *valeurs*}

Sortie : émettre des lignes $(j - l_j - i - C_j)$ avec $|l_j| = \mathcal{N}$

1: *couleur_actuelle* $\longleftarrow$ *lignevaleurs*[k][$\mathcal{N}+3$] {*valeurs*[k][$\mathcal{N}+3$] correspond à la couleur du nœud j pour le k^{ith} élément du vecteur *valeurs*}
2: **Si** *couleur_actuelle* $= C_j$ **Alors**
3: *liste_de_vote* $\longleftarrow$ [] {vecteur pour les $\mathcal{N}$ propagations de labels}
4: *nouvelle_ligne_de_labels* $\longleftarrow$ [] {vecteur pour les $\mathcal{N}$ propagations de labels}
5: **Pour** $w = 0$ à $\mathcal{N}$ par pas de 1 **Faire**
6: **Pour** $k = 0$ à $Taille(values)$ par pas de 1 **Faire**
7: {utilisation d'une double boucle *Pour* pour permettre de faire le vote colonne par colonne, pour les $\mathcal{N}$ propagations de labels différentes}
8: liste_de_vote.ajouter(*values*[w][k]) {*valeurs*[w][k] réfère au label du k^{th} voisin du nœud j de la w^{th} propagation de labels}
9: **Fin Pour**
10: *vote_majoritaire* $\longleftarrow$ $\arg\max_l |N^l(j)|$ utilisant *liste_de_vote*
11: *ligne_de_nouveaux_.ajouter*(*vote_majoritaire*)
12: **Fin Pour**
13: **Pour** $k = 0$ à $Taille(valeurs)$ par un pas de 1 **Faire**
14: Emettre(j - *nouvelle_ligne_de_labels* - *valeurs*[k][$\mathcal{N}$+2] - couleur_actuelle) {*valeurs*[k][$\mathcal{N}$ + 2] est le nœud transmetteur pour les labels, }
15: Emettre(*valeurs*[k][$\mathcal{N}$+2]$-$*nouvelle_ligne_de_labels*$-$*couleur_actuelle*$-$"$*$") {le signe "$*$" sera utilisé pour la mise à jour des labels du voisinage du nœud transmettant la nouvelle information. Elle sera à la position ($\mathcal{N}+2$) de la ligne émise}
16: **Fin Pour**
17: **Sinon**
18: Emettre($j - l_i - i - C_j$)
19: **Fin Si**

Algorithme 13 `UpdateReduce`

Entrée : $\mathcal{N}$ le nombre de lancements, Reduce(Clé : j, Itérateur valeurs : lignes avec propagation de labels, coloration et nœuds à mettre à jour [[$(l_i - i - C_j)$],[$l_i - i -$ " $*$ "],...]

Sortie : émettre les lignes $j - l_j - i - C_j$ (with $|l_j| = \mathcal{N}$)

1: **Pour** $w = 0$ à Taille(valeur) par pas de 1 **Faire**
2: **Si** $valeurs[w][\mathcal{N}+1] =$ " $*$ " **Alors**
3: $nœud\_a\_modifier \longleftarrow valeurs[w][\mathcal{N}]$
4: $nouveaux\_labels \longleftarrow valeurs[w]$
5: **Pour** $k = 0$ à Taille(valeurs) par pas de 1 **Faire**
6: **Si** non(" $*$ " dans $values[k]$) et ($nœud\_a\_modifier = valeurs[w][\mathcal{N}]$) **Alors**
7: $values[k] \longleftarrow nouveaux\_labels$ {Phase de mise à jour}
8: **Sinon**
9: Ne rien faire
10: **Fin Si**
11: **Fin Pour**
12: **Sinon**
13: Ne rien faire
14: **Fin Si**
15: **Fin Pour**
16: **Pour** $k = 0$ à Taille($valeurs$) par pas de 1 **Faire**
17: **Si** pas(" $*$ " dans $valeurs[k]$) **Alors**
18: Emettre($j -$ $valeurs[k]$) { équivalent à $j - nouveaux\_labels - i - C_j$ (avec $|nouveaux\_labels| = \mathcal{N}$)}
19: **Sinon**
20: Ne rien faire
21: **Fin Si**
22: **Fin Pour**

Algorithme 14 `CommunityReduce`

Entrée : $\mathcal{N}$ le nombre de lancements, Reduce(Clé : j, Itérateur *valeurs* : lignes avec propagation de labels et coloration [[$(l_i - i - C_j)$],...])

Sortie : émission de lignes $i - l_i$ (avec $|l_i| = \mathcal{N}$)

1: $couleur\_actuelle \longleftarrow ligne[\mathcal{N}+3]$ {$ligne[\mathcal{N}+3]$ correspond à la couleur du nœud j}
2: $liste\_des\_votes \longleftarrow$ [] {vecteur pour les $\mathcal{N}$ propagations de labels}
3: $ligne\_de\_nouveaux\_labels \longleftarrow$ [] {vecteur pour les $\mathcal{N}$ propagations de labels}
4: **Pour** $w = 0$ à $\mathcal{N}$ par pas de 1 **Faire**
5: **Pour** $k = 0$ à $Taille(values)$ par pas de 1 **Faire**
6: liste_des_votes.ajouter($valeurs$[w][k]) {$valeurs$[w][k] réfère au label du k^{ieme} voisin du nœud j à la w^{ieme} propagation de label}
7: **Fin Pour**
8: $vote\_majoritaire \longleftarrow \arg\max_l |N^l(j)|$ avecg $liste\_des\_votes$
9: $ligne\_de\_nouveaux\_labels.ajouter(vote\_majoritaire)$
10: **Fin Pour**
11: **Pour** $k = 0$ à $Taille(valeurs)$ par pas de 1 **Faire**
12: **Emettre**($valeurs[k][\mathcal{N}+2]$ - $ligne\_de\_nouveaux\_labels$) {Un dernier vote est fait, valeurs[k][$\mathcal{N}+2$] est équivalent au nœud i, ainsi (i - $ligne\_de\_nouveaux\_labels$)}
13: **Fin Pour**

Algorithme 15 `OccurrenceMatrixMap`

Entrée : $\mathcal{N}$ le nombre de lancements, Map(Clé : nom du fichier , valeur : fichier d'arêtes avec propagation de labels i_l_i)

Sortie : émission de lignes $k - i - l_i^k$ (pour la k^{th} propagation de labels du nœud i)

1: $l_j \longleftarrow$ [] {vecteur pour les $\mathcal{N}$ propagations de labels}
2: **Pour** line in Input **Faire**
3: **Pour** k = 0 to $\mathcal{N}$ **Faire**
4: **Emettre**($k - i - l_i^k$) {label(i) signifie le label du nœud i }
5: **Fin Pour**
6: **Fin Pour**

Algorithme 16 `GPrimAlphaReduce` $-\alpha$

Entrée : $\mathcal{N}$ le nombre de lancements, Reduce(Clé : nœud i, Valeur : Itérateur *valeurs* $[[i_1, 1], [i_2, 1], ..., [i_K, 1]]$, réel : α)
Sortie : émission de lignes (nœud $i-$ nœud $j - \alpha_{ij}$) avec $\alpha_{ij} \geq \alpha$ et $i > j$ depuis que la matrice est symétrique

1: **Pour** $v \in$ *valeurs* **Faire**
2: $Somme \longleftarrow 0$
3: $NbOcc \longleftarrow$ nombre d'occurrences du nœud v dans *valeurs*
4: **Si** $\frac{NbOcc}{\mathcal{N}} \geq \alpha$ **Alors**
5: $\alpha_{iv} \longleftarrow \frac{NbOcc}{\mathcal{N}}$
6: **Emettre** $(i, v, \alpha_{i,v})$
7: **Sinon**
8: Ne rien faire
9: **Fin Si**
10: **Fin Pour**

Si k est le nombre de couleurs, les votes auront lieu pour les nœuds ayant la couleur C_1, suivis d'une mise à jour (permettant la mise à jour des labels pour les nœuds émetteurs ou concernés par ces changements de labels). C'est ensuite pour les nœuds ayant la couleur C_2 que le vote s'effectue, suivi également d'une mise à jour. Le processus continue jusqu'à parvenir à la couleur C_k.
`VoteMap` consiste à émettre $j - l_i - i - C_j$, aucune opération spécifique n'est effectuée. La phase de shuffling permet de trier les données selon le nœud j, qui sera mis à jour si C_j est la couleur du nœud actuel considéré. Toutes les lignes avec la même clé j vont dans le même reducer. `VoteReduce` prend comme clé le nœud j et comme valeur, des vecteurs de labels des nœuds qui sont voisins du nœud j.

`VoteReduce` fait un vote pour le nœud j si la couleur associée à j est la couleur actuelle. Cela aura pour conséquence la mise à jour du vecteur de labels. Pour chacune des $\mathcal{N}$ propagations de labels, le vote majoritaire est fait suivant le LPA standard, colonne par colonne, par utilisation des vecteurs de labels des nœuds qui sont voisins du nœud j, dans ce cas $l_{j_1}, l_{j_2}, ..., l_{j_k}$ (si le nœud j a k voisins). Le reducer émet pour le nœud j son nouveau vecteur de labels qui est reçu de ses voisins, $j - l_i^{nouveau} - i - C_j$ où $l_i^{nouveau}$ est le nouveau vecteur de labels de i qui est transmis à j, résultant du vote. Une information de mise à jour est aussi émise pour les nœuds qui sont concernés par le nouveau label du nœud j, en émettant $i - l_i^{nouveau} - j - *$, $*$ permettant de connaître l'information de mise à jour et i étant le nœud dont le vecteur de propagation de labels sera mis à jour pour le prochain job. Nous donnons un petit exemple avec l'exemple du triangle à Fig. 4.4.

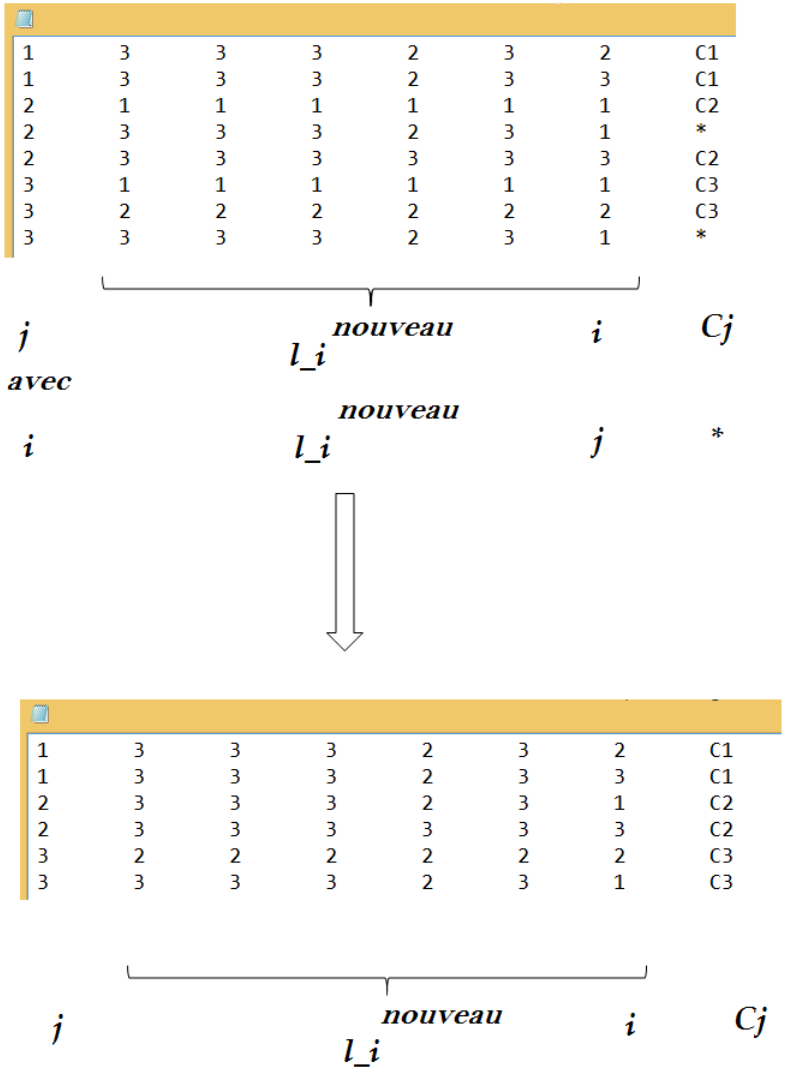


Figure 4.4 – a) Les résultats de `VoteMap` avec `VoteReduce`, b) `UpdateMap` avec `UpdateReduce`

Le job comprenant `UpdateMap` Algorithme 11 et `UpdateReduce`, Algorithme 13 effectue la mise à jour d'information des nœuds qui sont affectés directement par les changements de labels du dernier `VoteReduce`. Pour savoir quels nœuds sont concernés, nous utilisons les lignes contenant le symbole "*", émis par `VoteReduce`.

`UpdateMap` : Cette fonction lit les données, et émet chaque ligne sans faire

de modification. Aucune opération spécifique n'est effectuée. La phase de shuffling permet de trier les données selon le nœud receveur j.
`UpdateReduce` : Cette fonction prend en paramètre les données ayant la même clé j, et comme valeur, un itérateur avec les différentes propagations de labels ($j - l_i^{nouveau} - i - C_j$ et $i - l_i^{nouveau} - j - *$). Pour chaque partie ayant la même clé, si la fonction détecte le symbole $*$, une mise à jour se fera sur les lignes sujettes à l'ancienne propagation de labels (des nœuds voisins ou des nœuds transmetteurs d'information). Ainsi, les lignes $j - l_i^{new} - i - *$ serviront à faire la mise à jour des lignes de la forme $i - l_i - j - C_i$.

La troisième étape consiste à donner à chaque nœud un vecteur de label, sans coloration. Cette étape est réalisée par le job comprenant `CommunityMap`, Algorithme 11, et `CommunityReduce`, Algorithme 14. Les résultats seront utilisés pour la phase de détection de cœurs.

`MapCommunity` : Cette fonction, Algorithme. 9, lit les données, et émet chaque ligne sans aucune modification. Aucune opération spécifique n'est effectuée. la phase de shuffling permet de trier les données sur le disque suivant le nœud receveur j. La signature de cette fonction est : `CommunityMap` (Clé : fichier $(j - l_i - i - C_j) \rightarrow$ (Clé : j, Valeur : (l_i, i, C_j))).

`CommunityReduce` : Cette fonction reçoit comme entrée toutes les lignes ayant la même clé j et leurs vecteurs de labels, les nœuds voisins et la couleur relative comme valeur. Pour chaque ligne j, un dernier vote est fait de telle sorte que le reducer émette chaque nœud avec son vecteur de labels associé : $j - l_i - i - C_i(\forall i \in V(j)) \rightarrow i - l_i^{nouveau}$. La signature de cette fonction est : `CommunityReduce`(Clé : j , Valeur : Itérateur $([[l_i - i - C_j], ...]) \rightarrow$ (Clé : i, Valeur : $(l_i^{nouveau})$)).

La quatrième partie consiste à remplir une matrice de co-fréquence, $P_{ij}^{\mathcal{N}}$, où la colonne i et la ligne j représentent le nombre de fois que les nœuds i et j sont ensemble durant les $\mathcal{N}$ différentes propagations de labels. Ce job consiste en `OccurrenceMatrixMap`, Algorithme 15 et `OccurrenceMatrixReduce`. Le mapper retourne pour chaque propagation de labels le nœud avec sa couleur associée et le reducer émet chaque paire de nœuds qui sont ensemble durant les $\mathcal{N}$ propagations de labels avec le chiffre "1".

`OccurrenceMatrixMap` prend comme entrée $i - l_i$ (le nœud et son vecteur associé de labels) et retourne $r_k - i - l_i^k$ où r_k est la k^{ieme} propagation de labels et l_i^k, le k^{ieme} label correspondant au nœud i. La phase de shuffling trie les données selon la clé r_k, soit la k^{ieme} propagation de labels. La signature de cette fonction est : `OccurrenceMatrixMap`(Cle : fichier $(i - l_i) \rightarrow$ (Clé : r_k,

Valeur : (i, l_i^k)). `OccurrenceMatrixReduce` prend en paramètre $r_k - i - l_i$ et émet $i - j - 1$, soit chaque paire de nœuds qui sont dans les mêmes communautés, propagation de labels après propagation de labels. La signature de cette fonction est : `OccurrenceMatrixReduce`(Cle : r_k , Valeur : Itérateur ($[[i, l_i^k], ...]$) $\rightarrow$ (Clé : (i, j), Valeur : 1)).

La cinquième partie consiste en `GPrimAlphaMap`, Algorithme 11 et `GPrimAlphaReduce`, Algorithme 16.

`GPrimAlphaMap` prend en entrée $i - j - 1$ et émet cette même ligne. Aucune opération spécifique n'est effectuée. La phase de shuffling permet de trier la ligne selon le nœud i. La signature de cette fonction est : `GPrimAlphaMap`(Cle : fichier $(i, j, 1) \rightarrow$ (Cle : i, j, Valeur : 1)).

`GPrimAlphaReduce` prend comme clé le nœud i, $i - j - 1$ et α comme valeur. α permet d'émettre les arêtes dont les nœuds ont une fréquence d'apparition commune dans les mêmes communautés supérieure ou égale à ce seuil. Cette fonction émet $i - j - \alpha_{ij}$ ou $\alpha_{ij} \geq \alpha$, où α_{ij} est la fréquence où les nœuds i et j se sont trouvés dans les mêmes communautés au cours des $\mathcal{N}$ différentes propagations de labels. La signature de cette fonction est : `GPrimAlphaReduce`-α (Clé : r_k , Valeur : Itérateur ($[[j, 1], ...]$) $\rightarrow$ (Clé : (i, j), Valeur : α_{ij})).

La dernière étape consiste à trouver les composantes connexes qui correspondent aux communautés. PEGASUS (Kang *et al.* (2009)), un outil d'analyse de réseaux sociaux destiné aux grands graphes et fondé sur le patron de conception MapReduce offre une implantation d'un algorithme de détection de composantes connexes.

On peut ainsi résumer les différentes étapes du Par-CDLP par l'algorithme regroupant la succession de jobs, Algorithme 17. On note qu'il est possible d'appliquer la simple propagation de label semi-synchrone (SLPH) sous Hadoop avec $\mathcal{N} = 1$ en terminant avec la fonction `CommunityReduce`.

4.1.3 Création d'un dendrogramme

Il est également possible de produire un dendrogramme avec notre modèle Hadoop en créant plusieurs jobs (`GPrimAlphaMap` et `GPrimAlphaReduce`) avec différentes valeurs de α. Il est ainsi possible de créer un intervalle avec un pas Δ, et plusieurs fichiers avec différentes valeurs de α, comme une suite numérique $(u_n)_{min \leq n \leq max}$ de telle sorte $u_{n+1} = u_n + \Delta$. Le dendrogramme entier peut être obtenu avec $u_0 = 0$ et $max = 1$. Si le niveau u_{n+1} est le même que le niveau u_n (même partitionnement, donnant les mêmes communautés au

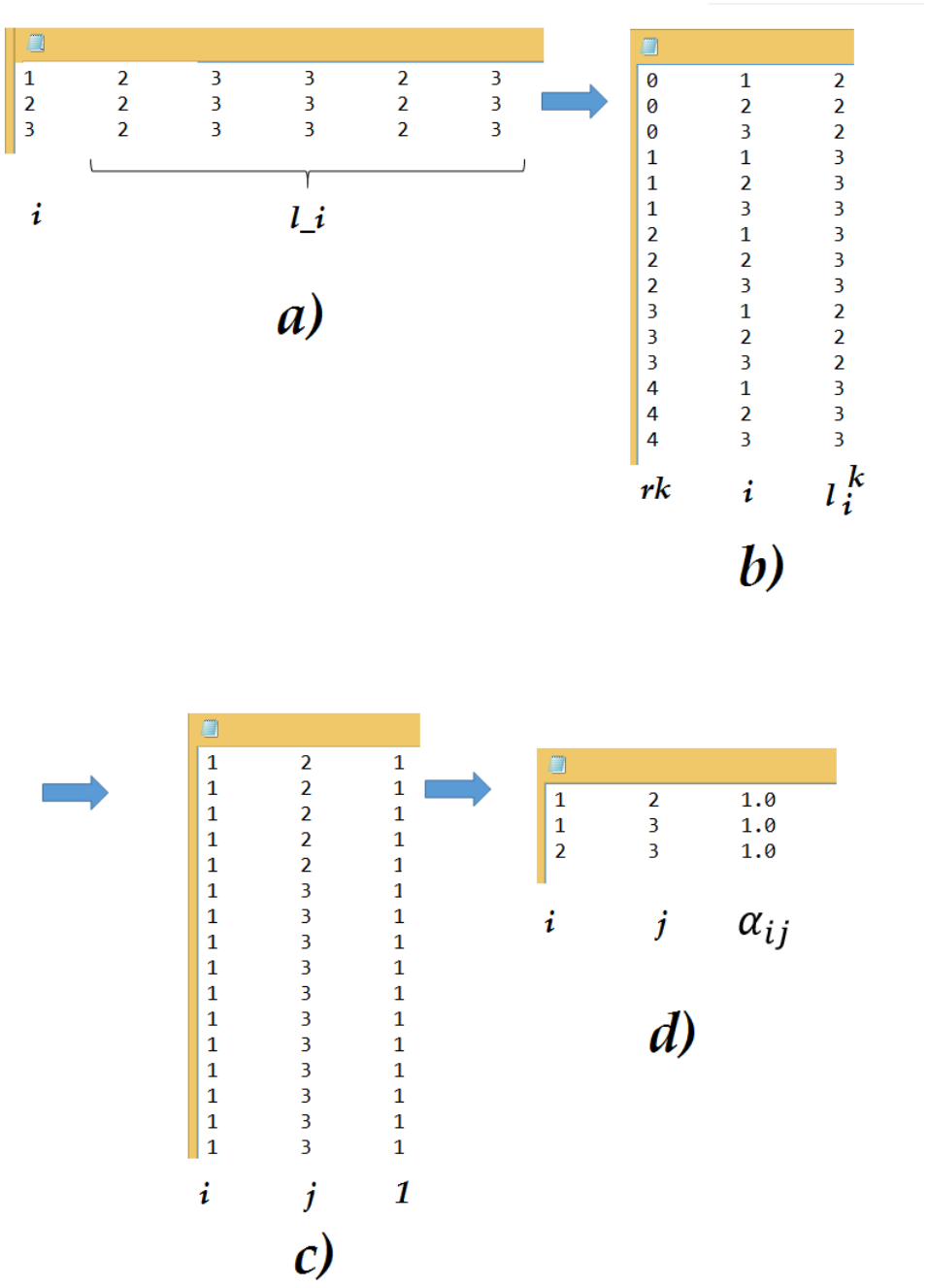


Figure 4.5 – a) CommunityMap et CommunityReduce b) OccurrenceMatrixMap c) GPrimAlphaMap d) GPrimAlphaReduce

Algorithme 17 Le CDLP

Entrée : Un graphe $G = (V, E)$, le seuil α, $\mathcal{N}$ le nombre de lancements, Δ le pas

Sortie : communautés de G

```
FirstReadMap-𝒩
FirstReadReduce
Pour i = 0 au nombre de couleur de G Faire
  VoteMap
  VoteReduce-C_i
  MapUpdate
  UpdateReduce
Fin Pour
CommunityMap
CommunityReduce
OccurrenceMatrixMap
OccurrenceMatrixReduce
GPrimAlphaMap
GPrimAlphaReduce
return La partition : P = {P_1, ..., P_C}.
```

nœud près), le niveau u_{n+1} n'est pas écrit. Plus faible sera le pas Δ, plus grand sera la taille de l'arbre.

4.2 Expérimentations

Nous expérimentons notre proposition algorithmique sur quatre grands graphes de la littérature qui sont Amazon, DBLP, You Tube et Live Journal pour étudier la qualité des communautés résultantes, la scalabilité de notre proposition algorithmique, le temps d'exécution et la masse des données générées et traitées. Pour nos expérimentations, nous avons un cluster Hadoop de cinq machines dont la description est au Tableau 4.1. Une étude comparative sera proposée avec certains algorithmes de la littérature.

4.2.1 Etude de la stabilisation

Comme nous l'avions vu au chapitre 2, notamment pour le CDLP, un certain nombre de propagations de labels suffit à stabiliser l'algorithme suivant la topologie du graphe. Cependant, cela ne rend pas l'algorithme totalement déterministe. Certaines oscillations peuvent apparaître, notamment sur les nœuds considérés comme chevauchants. Sur l'exemple de Zachary, nous avions vu qu'il suffisait de 55 propagations de labels pour stabiliser la méthode, avec une fluctuation sur le nœud 10 qui est entre les deux principales communautés. Ainsi,

Cluster Hadoop (5 machines Acer Predator G3-605)	
Nb. de cœurs	4 (Cor TM i7)
Nb. de threads	12
Fréquence de base	3.00 GHz
Fréquence Turbo maxi	3.60 GHz
Hadoop	version 2.7.2
Connection ethernet	1 Gigabits entre les machines
Capacité mémoire maxi	1 TO

Tableau 4.1 – Caractéristique du cluster Hadoop

des oscillations faisaient que nous avions 1 ou 2 communautés pour $\alpha \leq 0.5$. Si l'on considère de grands graphes de terrain tels qu'Amazon, DBLP, You-Tube Ou Live Journal, des oscillations peuvent également avoir lieu. C'est ce que nous avons pu observer et qui a rendu l'étude de la stabilisation très complexe. On ne peut pas simplement se fonder comme indicateur sur le nombre de communautés, la fluctuation due à certains nœuds rend l'étude très difficile. Il a été montré qu'il existait des milliers de structures communautaires dans les réseaux que nous avons cités. En ayant observé de très nombreuses oscillations, nous ne sommes pas dans la capacité de conclure quant au nombre de propagations de labels nécessaire. C'est actuellement une voie de recherche sur laquelle nous portons notre attention. C'est en ce sens que nous avons décidé de poser pour la suite de nos expérimentations $\mathcal{N} = 200$. Nous proposons également une étude sur la propagation de label semi-synchrone sous Hadoop (SLPH avec $\mathcal{N} = 1$).

Nous donnons un exemple d'expérimentation où la seule observation tangible est que l'augmentation du nombre de propagations de labels permet de réduire le nombre de communautés jusqu'à des oscillations qui ne permettent pas de conclure. Pour le calcul du NMI et du F_1 scores, nous utilisons la connaissance des vraies communautés, leur top 5000, disponible sur SNAP à l'adresse $http://snap.stanford.edu$.

4.2.2 Etude sur de grands graphes de terrains

Etude sur Amazon

Le nombre de couleurs détectées est de 24 pour faire marcher la propagation de labels semi-synchrone, *Tableau 4.2*. La distribution des degrés moyens selon les couleurs est également donnée au Tableau 4.2.

La première analyse traite de la distribution des couleurs. Il y a 24 couleurs détectées sur le graphe d'Amazon. Nous voyons que 48.8% des nœuds ont la même couleur, c'est-à-dire qu'ils mettront à jour leurs propres vecteurs de labels en même temps durant le processus Hadoop. Nous observons également que le pourcentage de nœuds par couleur n'est pas uniformément distribué. Les couleurs C_1,C_2,C_3,C_4,C_5 et C_{23} représentent 86.5206% des nœuds du graphe alors

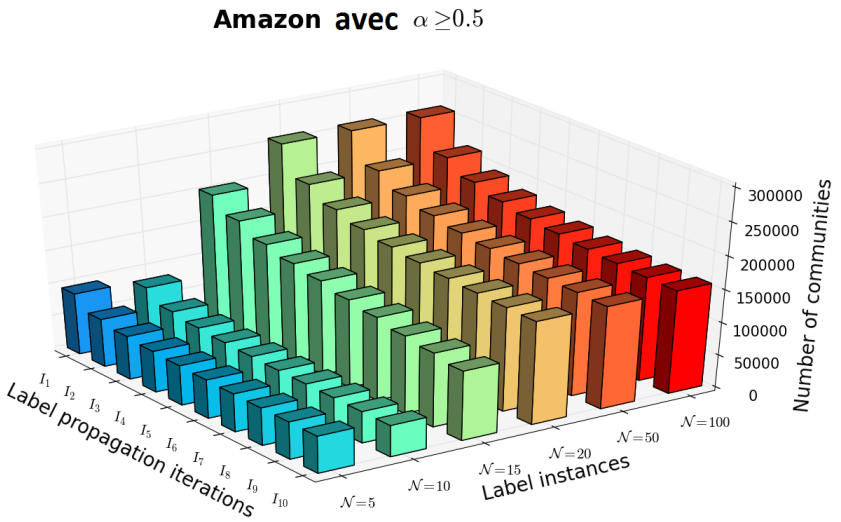


Figure 4.6 – Expérimentation sur la stabilisation du Par-CDLP avec Amazon et $\alpha \geq 0.5$

Coloration de graphe sur Amazon			
Color	*dist*	*Color*	*dist*
C_1	48.449 %	C_{13}	0.117 %
C_2	4.0476 %	C_{14}	0.106 %
C_3	16.003 %	C_{15}	0.035 %
C_4	4.524 %	C_{16}	0.030 %
C_5	7.987 %	C_{17}	0.008 %
C_6	3.856 %	C_{18}	0.007 %
C_7	3.659 %	C_{19}	0.002 %
C_8	2.420 %	C_{20}	0.001 %
C_9	1.369 %	C_{21}	0.0008 %
C_{10}	1.065 %	C_{22}	0.0005 %
C_{11}	0.429 %	C_{23}	5.5142 %
C_{12}	0.361 %	C_{24}	0.0003 %

Tableau 4.2 – Distribution des tailles (dist) pour chaque groupe ayant une couleur spécifique C_i

que C_{13},C_{14},C_{15},C_{16}, C_{17},C_{18}, C_{19},C_{20},C_{21},C_{22},C_{23} et C_{24} ne représentent que 0.3076% du total des nœuds du graphe.

Notre première étude porte sur le SLPH, c'est-à-dire notre proposition Hadoop avec $\mathcal{N} = 1$. Nous avons fait marcher le SLPH itération après itération, 10 fois, *Tableau 4.3*. L'objectif étant de voir si la propagation de label s'effectue de manière correcte itération après itération. Les observations nous montrent

que les communautés deviennent de plus en plus importantes itération après itération. La première propagation de label (I_1) donne $85,913\%$ de communautés avec une taille comprise entre 1 et 10 nœuds. A la dixième propagation de labels, nous avons $57,0532\%$ de communautés dont la taille est comprise entre 1 et 10. L'algorithme est capable de détecter des communautés de différentes tailles, petites et plus grandes. A I_{10}, $1,4774\%$ des communautés ont une taille comprise entre 41 et 50, alors que $0,3776\%$ des communautés ont une taille supérieure à 100. La figure 4.7 montre l'existence d'une convergence après 10 itérations. La conductance diminue parallèlement à la diminution constante du nombre de communautés, dont les densités augmentent. C'est en ce sens que nous proposons d'augmenter le nombre d'itérations à 15, pour une meilleure qualité de partitionnement.

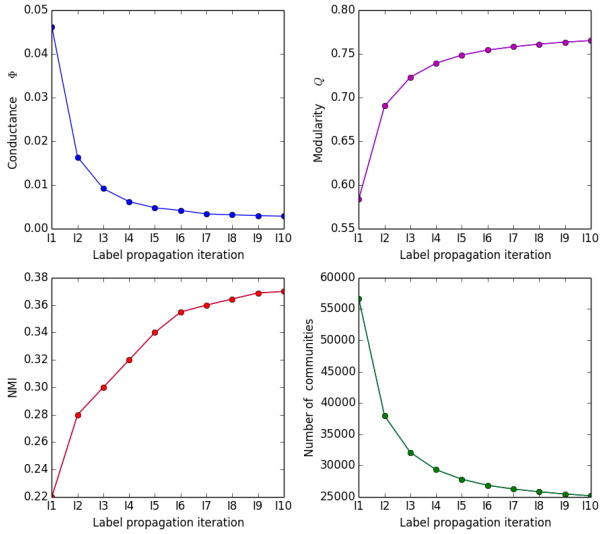


Figure 4.7 – Propagation de labels semi-synchrone sur Hadoop (SLPH)avec $\mathcal{N} = 1$, itération après itération.

En observant le Tableau 4.4, nous voyons que le seuil α joue bien un rôle sur la taille des communautés et la qualité qui en résulte. Plus α est important, plus le nombre de communautés augmente.

Pour $\mathcal{N} = 100$, Tableau 4.5, avec $\alpha \geq 0,4$, $73,081\%$ des communautés ont une taille comprise entre 1 et 10 nœuds alors qu'avec $\alpha \geq 0,6$, ce pourcentage s'élève à $89,057\%$. Néanmoins, certaines grandes communautés sont trouvées pour chaque valeur de α, montrant la robustesse de certaines communautés. Avec $\alpha \geq 0,4$, $1,012\%$ des communautés ont une taille supérieure à 100. Appliquer une matrice de co-fréquence diminue fortement la taille des communautés. Avec $\alpha \geq 0,8$, seulement $0,0018\%$ des communautés ont une taille supérieure à

Distribution des tailles de communautés avec SLPH sur Amazon					
Tailles	I_1	I_2	I_3	I_4	I_5
1-10	85.913 %	72.8953 %	66.7653 %	63.1801 %	61.1225 %
11-20	11.1707 %	19.5767 %	22.7308 %	24.4647 %	25.2975 %
21-30	2.046 %	4.7709 %	6.369 %	7.2787 %	8.0646 %
31-40	0.5455 %	1.5736 %	2.312 %	2.6763 %	2.7505 %
41-50	0.1854 %	0.5957 %	0.8725 %	1.1455 %	1.3159 %
51-60	0.0759 %	0.3163 %	0.4518 %	0.5728 %	0.6652 %
61-70	0.0388 %	0.1476 %	0.243 %	0.2864 %	0.302 %
71-80	0.000 %	0.000 %	0.000 %	0.000 %	0.000 %
81-90	0.0106 %	0.029 %	0.0623 %	0.133 %	0.1546 %
91-100	0.0018 %	0.0316 %	0.0592 %	0.0545 %	0.0791 %
> 100	0.0124 %	0.0604 %	0.1276 %	0.2009 %	0.2411 %
Tailles	I_6	I_7	I_8	I_9	I_{10}
1-10	59.5642 %	58.6458 %	58.0986 %	57.4829 %	57.0532 %
11-20	26.0335 %	26.5051 %	26.63 %	26.9699 %	27.1485 %
21-30	8.4246 %	8.52 %	8.763 %	8.9127 %	9.0191 %
31-40	2.9832 %	3.1702 %	3.1534 %	3.2328 %	3.336 %
41-50	1.3892 %	1.3794 %	1.5612 %	1.5005 %	1.4774 %
51-60	0.6965 %	0.7964 %	0.7399 %	0.8131 %	0.8062 %
61-70	0.3538 %	0.3658 %	0.3951 %	0.3967 %	0.4527 %
71-80	0.000 %	0.000 %	0.000 %	0.000 %	0.000 %
81-90	0.1713 %	0.202 %	0.2014 %	0.2043 %	0.2065 %
91-100	0.108 %	0.0953 %	0.1162 %	0.1139 %	0.1152 %
> 100	0.268 %	0.3085 %	0.3255 %	0.4084 %	0.3776 %

Tableau 4.3 – Distribution des tailles des communautés SLPH ($\mathcal{N} = 1$)

CDLP on Amazon (I_{15} and $\mathcal{N} = 100$)					
	$\alpha \geq 0.2$	$\alpha \geq 0.3$	$\alpha \geq 0.4$	$\alpha \geq 0.5$	$\alpha \geq 0.6$
Q	0.5559	0.5768	0.4789	0.3522	0.3248
ϕ	0.1176	0.1150	0.2160	0.3552	0.4125
NMI	0.3604	0.377456	0.359236	0.328038	0.3015
F_1	0.4225	0.433952	0.4240019	0.404496	0.3849
#	60077	69628	85801	101821	113972

Tableau 4.4 – Scores des mesures supervisées et non supervisées

100. Plus α augmente, plus la taille des communautés diminue. Le SLPH trouve approximativement 25 000 communautés.

L'analyse des mesures supervisées et non supervisées montre que la méthode de détection de cœurs donne de relativement bons résultats mais reste équivalente en termes de qualité au SLPH. Cela vient du fait que les structures communautaires du graphe Amazon sont de très petite taille.

CDLP sur Amazon (distribution des tailles des communautés) (I_{10} and $\mathcal{N} = 20$)					
Tailles	$\alpha \geq 0.4$	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$
1-10	73.081	82.145%	89.057 %	94.4227 %	97.729 %
11-20	12.057	10.257 %	7.303 %	4.2461 %	1.9291 %
21-30	7.548	5.251 %	1.9563 %	0.8188 %	0.2488 %
31-40	3.314	1.258 %	0.7863 %	0.2952 %	0.0611 %
41-50	1.369	0.554 %	0.3535 %	0.0984 %	0.0167 %
51-60	0.845	0.2748 %	0.1775 %	0.0594 %	0.009 %
61-70	0.145	0.082 %	0.1125 %	0.0297 %	0.0026 %
71-80	0.378	0.0240 %	0.0 %	0.0 %	0.0 %
81-90	0.024	0.0345 %	0.0563 %	0.0084 %	0.0013 %
91-100	0.1458	0.0347 %	0.0476 %	0.0046 %	0.0006 %
101-200	0.275	0.062%	0.1239 %	0.014 %	0.0012 %
201-300	0.351	0.120	0.0186 %	0.0009 %	0.0 %
>300	0.386	0.0232	0.0042 %	0.0012 %	0.0006 %

Tableau 4.5 – Distribution des tailles de communautés du CDLP et de SLPH

Nous donnons quelques exemples d'éléments issus des communautés que Par-CDLP a détectées, Tableau 4.6.

Exemples de communautés détectées		
Communautés	articles	genre et thème
Agatha Christie	The Man in the Brown Suit (St. Martin's Minotaur Mysteries)	enquête policière
	Lord Edgware Dies	enquête policière
	Poirot Investigates	enquête policière
	Murder on the Orient Express : A Hercule Poirot Mystery	enquête policière
	4 : 50 from Paddington Miss Marple Mysteries	enquête policière
Judith Philips	Southwestern Landscaping with Native Plants	jardinage (général)
	New Mexico Gardener's Guide (Gardener's Guides)	jardinage (désert)
	New Mexico Gardener's Guide (Jul 3, 2001)	jardinage (désert)
	Natural by Design : Beauty and Balance in Southwest Gardens	jardinage (sud-ouest)
	Plants for Natural Gardens : Southwestern Native, Adaptive Trees, Shrubs, Wildflowers, Grasses	jardinage (Texas et Californie)
Kelli Dolecek	Month-To-Month Gardening, New Mexico	jardinage (désert)

Tableau 4.6 – Exemple de communautés détectées sur Amazon

Etude sur DBLP

CDLP sur DBLP (I_{10} et $\mathcal{N} = 100$)						
Tailles	$\alpha \geq 0.3$	$\alpha \geq 0.4$	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	SLPH
Q	0.6424	0.6979	0.7104	0.6680	0.6273	0.6546
ϕ	0.0361	0.0523	0.00647	0.025	0.045	0.0612
NMI	0.1948	0.2202	0.260057	0.246206	0.2364	0.2241
F_1	0.267489	0.304575	0.373641	0.367221	0.345877	0.325454
#	24771	28660	36218	35439	40521	45276

Tableau 4.7 – Scores des mesures supervisées et non supervisées

Distribution des tailles de communautés, CDLP sur DBLP (I_{20} et $\mathcal{N} = 100$)					
Tailles	$\alpha \geq 0.5$	$\alpha \geq 0.6$	$\alpha \geq 0.7$	$\alpha \geq 0.8$	SLPH
1-10	79.300 %	85.6181 %	90.2142 %	93.4246 %	69.8484 %
11-20	13.400 %	10.0392 %	7.2451 %	5.1026 %	19.8401 %
21-30	3.800 %	2.488 %	1.5352 %	0.921 %	5.9319 %
31-40	1.500 %	0.8986 %	0.5514 %	0.315 %	2.2757 %
41-50	0.800 %	0.4176 %	0.2162 %	0.1235 %	0.9611 %
51-60	0.400 %	0.2197 %	0.1097 %	0.0403 %	0.4954 %
61-70	0.300 %	0.1306 %	0.0494 %	0.029 %	0.284 %
71-80	0.000 %	0.0 %	0.0 %	0.0 %	0.0 %
81-90	0.100 %	0.0455 %	0.0154 %	0.0113 %	0.0727 %
91-100	0.100 %	0.0238 %	0.0108 %	0.0063 %	0.0892 %
101-200	0.300 %	0.2701 %	0.0477 %	0.0293 %	0.1749 %
201-300	0.0929 %	0.0785 %	0.0517 %	0.0428 %	0.0014
> 300	0.000 %	0.0132 %	0.008 %	0.00 %	0.00 %

Tableau 4.8 – Distribution des tailles des communautés avec Par-CDLP

L'algorithme de coloration trouve pour le graphe DBLP 116 couleurs. La distribution des tailles des groupes de nœuds suivant les couleurs n'est pas uniformément répartie. 7 couleurs réprésentent près de $90,377\%$. Certaines couleurs ne réprésentent que $0,0003\%$ de la population totale.

Le SLPH trouve 24469 communautés avec une modularité de 0.6894 et un NMI de 0.218794. La majorité des communautés détectées sont de taille comprise entre 1 et 10 (69.8484 %). Cependant, de grandes communautés sont détectées avec plus de 0.1895% des communautés ayant une taille supérieure à 100 nœuds.

Nous observons au tableau 4.7 que la qualité des communautés détectées par Par-CDLP, notamment pour $\alpha \geq \{0.5, 0.6, 0.7, 0.8\}$, est meilleure qu'avec le SLPH. Par-CDLP obtient un NMI de 0.260057 avec $\alpha \geq 0.5$. La méthode à base de détection de cœurs permet d'améliorer la qualité de partitionnement. Pour $\alpha \geq 0.8$, le nombre de communautés avec Par-CDLP devient très important. Comme SLPH, Par-CDLP détecte quelques grandes communautés qui

disparaissent avec l'augmentation de α.
Nous donnons quelques exemples de communautés détectées avec plusieurs valeurs de α, *Tableau 4.9*. (Il ne s'agit que d'une partie des communautés).

Exemples de communautés détectées sur DBLP ($n = 317080$ et $m = 1049866$)		
Auteurs	communautés	genre et thème
Maria Malek ($\alpha \geq 0.3$)	"Dominique Laurent" "Dalia Sulieman" "Hubert Kadima" "Rushed Kanawati" "Sylvie Salotti" "Farida Zehraoui" "Vincent Rialle" "Ahmad A. Kardan" "AHM Ragab" "M Ebrahimi" "DR Millen" "Nicola Spyratos" etc.	Recommandation sématique et sociale Recommandation sémantique et sociale Recommandation sémantique et sociale COBRA recommandation système multi-agent
Maria Malek ($\alpha \geq 0.85$)	"Rushed Kanawati"	COBRA
Marc Zolghadri ($\alpha \geq 0.3$)	"Rachid Chelouah" "Citlalih Gutierrez Estrada" "Bruno Vallespir" "Yan Liu" "Salah Zouggar" "Stéphane Brunel" "Philippe Girard" "Claude Olivier" "P Agarwal" "M Sahai" "V Mishra" "Claude Baron" etc.	Meta-heuristics Power-based supplier Power-based supplier Power-based supplier

Tableau 4.9 – Exemple de communautés détectées

Etude comparative

Nous proposons une étude comparative portant sur la qualité de partitionnement. Les algorithmes pour cette étude sont la méthode de Louvain (Blondel *et al.* (2008)), WalkTrap (Pons et Latapy (2006)), OSLOM (Lancichinetti *et al.* (2011)), Infomap (Rosvall et Bergstrom (2008)), Bigclam (Yang et Leskovec (2013)), SCD (Prat-Pérez *et al.* (2014)), LPA (Raghavan *et al.* (2007)) et SLPA (Xie *et al.* (2011)) sur quatre grands graphes, Amazon, DBLP, You Tube et Live Journal.

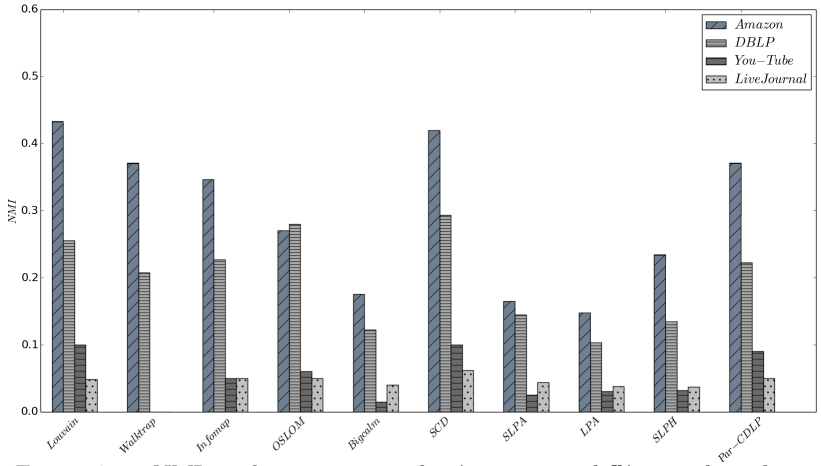


Figure 4.8 – NMI sur les quatre grands réseaux avec différents algorithmes

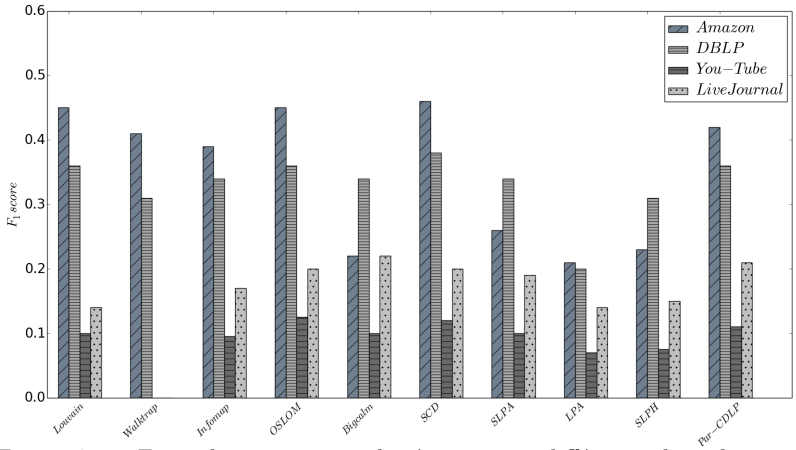


Figure 4.9 – F_1 sur les quatre grands réseaux avec différents algorithmes

Les premières observations des Figures 4.8 et 4.9 montrent que notre solution est compétitive par rapport aux solutions issues de la litérature. On observe

que le CDLP donne de meilleurs résultats que les autres méthodes à base de propagation de labels. SLPH est presque équivalent au LPA sur you Tube et live Journal, mais donne cependant de meilleurs résultats en termes de qualité sur Amazon ou DBLP. Cependant, d'autres méthodes surpassent CDLP comme Louvain ou SCD en termes de qualité de partitionnement.

4.2.3 Etude volumétrique traitée par le HDFS

Nous étudions la quantité de données produites par chaque job. En effet, certaines solutions de cloud computing comme "Google Cloud Paltform" *(https ://cloud.google.com/)* ou "Amazon EMR" *(https ://aws.amazon.com)* facturent selon la quantité de données stockée. Sur la Figure 4.10, nous récapitulons chaque job, avec les mappers et reducers associés.

Jobs MapReduce		
job_1	`FirstReadMap`	`FirstReadReduce`
job_2	`VoteMap`	`VoteReduce`$-C_i$
job_3	`MapUpdate`	`UpdateReduce`
job_4	`MapCommunity`	`ReduceCommunity`
job_5	`MapOccurrenceMatrix`	`ReduceOccurrenceMatrix`
job_6	`MapGPrimAlpha`	`ReduceGPrimAlpha`

Tableau 4.10 – jobs MapReduce pour le Par-CDLP

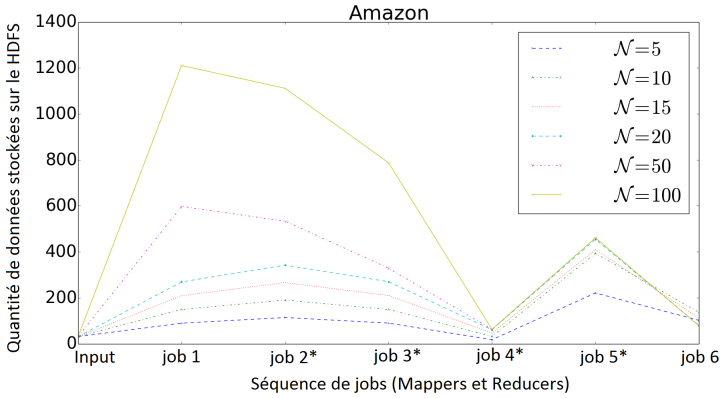


Figure 4.10 – Quantité de données produite par les jobs et stockée par le HDFS en fonction du nombre de propagations de labels lancées en parallèle, "*" signifie que le calcul est une moyenne, $\alpha \geq 0.1$.

Initialement, nous utilisons 80 reducers pour une simple machine. Le nombre de reducers sera multiplié par deux à chaque fois qu'une nouvelle machine sera ajoutée. Le temps nécessaire, Figure 4.11, pour faire marcher la propagation

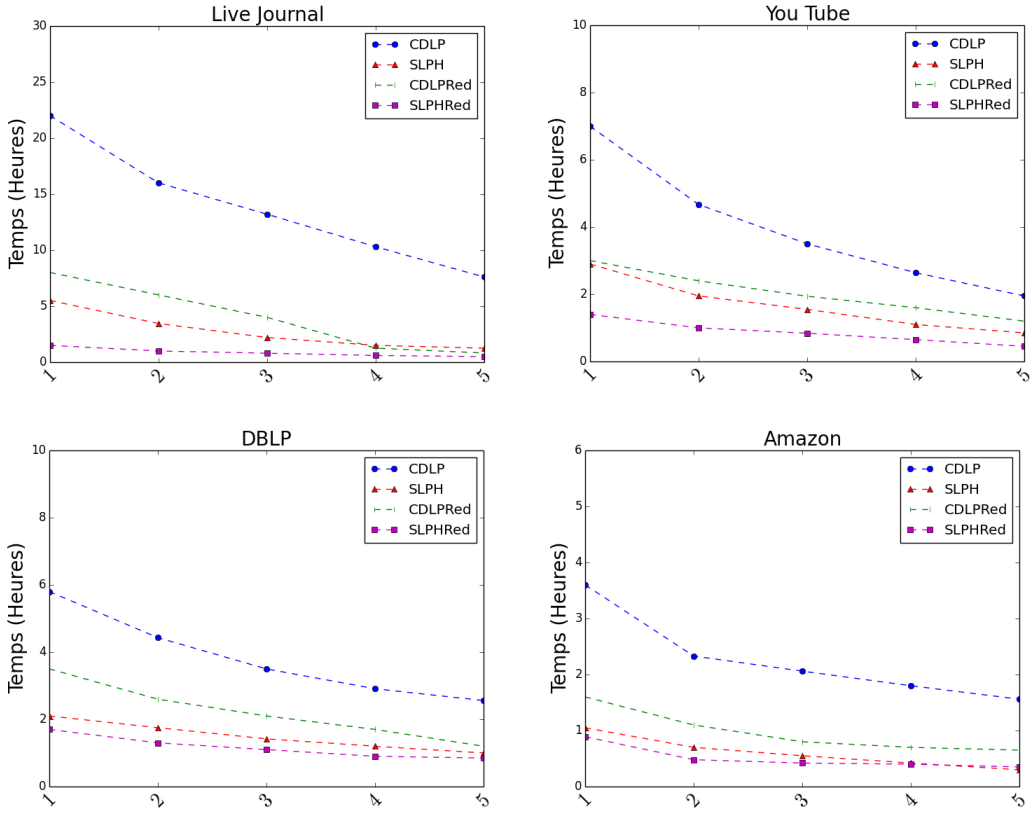


Figure 4.11 – Temps d'exécution (en heures), l'axe des abcisses représente le nombre de machines fonctionnant en parallèle

de labels semi-synchrone sur Amazon, de l'étape de coloration à l'étape de la détection des composantes connexes avec $\mathcal{N} = 100$ est d'environ deux heures pour une simple machine (avec une version Hadoop standalone) alors qu'il n'est que de 35 minutes pour le SLPH. Il est donc intéressant de pouvoir utiliser un cluster de machines pour tenter de réduire le temps d'exécution. Avec l'augmentation des machines, le temps d'exécution diminue pour arriver à 35 minutes, toujours avec $\mathcal{N} = 100$.

La quantité de données stockée est représentée à la Figure 4.10. Par défaut, le HDFS réplique chaque bloc de données produit 3 fois sur son cluster. Nous avons volontairement supprimé cette fonctionnalité pour mieux rendre compte de la quantité de données produite par le logiciel. Nous rappelons que les noms des jobs sont au Tableau 4.10. Pour les jobs $2, 3, 4$ et 5, nous donnons la quantité de données moyenne produite par une couleur depuis que nous procédons à la

destruction des données intermédiaires, avec un script de nettoyage de données. Nous observons un facteur de proportionnalité en fonction de $\mathcal{N}$. Les jobs $1, 2$ et 3 produisent les plus grandes quantités de données. Pour le job 5 et pour chaque $\mathcal{N}$, la quantité de données est presque la même depuis que ce job associe à chaque nœud sa communauté.

Nous avons également observé que la valeur de α n'avait aucune influence sur le temps et sur la quantité de données produites par les jobs 1 à 5. Cela vient du fait que ce paramètre n'intervient pas dans le processus algorithmique. Il intervient cependant au job 6 où la volumétrie et le temps d'exécution fluctueront en fonction de la valeur α. La valeur de α aura pour conséquence une fluctuation du temps de calcul des composantes connexes, qui représentent nos communautés. Un α produisant plus d'arêtes demandera un temps de calcul pour les composantes connexes plus important.

Sur la Figure 4.10, le volume de données a été calculé avec $\alpha \geq 0,1$. C'est le job 1 qui produit la plus importante quantité de données depuis qu'il a fallu instancier un vecteur de labels de taille $\mathcal{N}$, $j - li - i - C_i$.

Nous avons, à titre expérimental, testé plusieurs valeurs de $\mathcal{N}$ et joué sur le nombre de couleurs pour le temps d'exécution, à la Figure 4.11. Le temps d'exécution de notre modèle sera fonction de $\mathcal{N}$, mais également du nombre de couleurs détectées par l'algorithme de coloration. Plus le nombre de couleurs est important, plus le temps d'exécution est important. Sur DBLP, nous avons près de $90,377\%$ des nœuds qui ne réprésentent que 7 couleurs alors que cela est du même ordre pour Amazon, avec une couverture de 90.355% de nœuds avec juste 7 couleurs. Cela sous entend que de nombreux jobs traiteront uniquement 10% des nœuds du graphe, avec un temps excessivement important dû à l'importance du nombre de couleurs. C'est en ce sens que nous lançons à la fois l'algorithme original et une version réduite (CDLPRED) comprenant exclusivement 7 couleurs plus 1 qui servira à la propagation des autres labels à la fin du processus.

En ce qui concerne Par-CDLP, augmenter le nombre de machines et le nombre de reducers diminue le temps d'exécution pour un $\mathcal{N}$ élevé. Ce n'est pas toujours le cas lorsque la quantité de données à traiter n'est pas très inportante comme pour SLPH. Nous voyons que les versions réduites vont beaucoup plus vite. En effet, certaines couleurs ne représentent que certains nœuds du graphe (comme la couleur C_{24} avec moins de $0,0003\%$ du total des nœuds du graphe). Il faut beaucoup de temps pour effectuer la propagation d'un nombre aussi faible de labels.

4.3 Conclusion portant sur les propositions liées au parallélisme et la distribution

Dans ce chapitre, nous avons décrit une implantation informatique possible parallèle et distribuée de la propagation de labels avec cœurs. Pour résoudre le problème de la propagation de labels asynchrone, nous avons proposé d'utiliser le principe de coloration, où des groupes de nœuds changent leurs labels en même temps suivant la propagation de labels habituelle, alors que d'autres attendent leur tour.

En utilisant comme critère la modularité, les expérimentations ont montré de meilleurs résultats en termes de qualité de partitionnement par rapport aux méthodes à base de propagation de labels comme le LPA, mais cependant moins bons que pour la méthode de Louvain.

La méthode proposée est relativement lente. Le temps d'exécution, que ce soit pour DBLP, You Tube ou live Journal peut nécessiter plusieurs heures. Le nombre de couleurs joue un rôle majeur sur le temps d'exécution. Par exemple, nous avons trouvé 24 couleurs sur le graphe d'Amazon alors que 6 couleurs représentent 86.52% des nœuds du graphe. Certaines couleurs ne représentent quant-à-elles que 0.0003% du total des nœuds du graphe, soit une dizaine de nœuds. Appliquer Par-CDLP sur un aussi petit groupe de nœuds requiert un temps très important. Ce constat fut établi pour tous les grands graphes qui furent utilisés pour les expérimentations. De plus, l'écriture sur disque des données requiert un temps qui peut être très important.

Nous étudions actuellement un moyen de réduire le temps d'exécution, comme fusionner des groupes de nœuds de couleurs différentes au risque de détériorer la qualité de partitionnement. Cette implémentation pourrait également être utile pour une version parallèle et distribuée destinée au chevauchement. Nous étudions également une version en mémoire RAM, notamment en utilisant Apache Spark.

Chapitre 5

Conclusion

Sommaire

Dans cette thèse, nous nous sommes intéressés aux problèmes de détections de communautés disjointes et chevauchantes, et à la scalabilité de nos méthodes en proposant une version Hadoop pour la propagation de labels avec détection de cœurs.

Le premier chapitre a permis de décrire trois grandes classes d'algorithmes en détection de communautés disjointes : les méthodes globales, locales et hybrides. Il a été observé que les méthodes locales, c'est-à-dire dont le point de départ est atomique (par le nœud), permettaient de traîter de plus grands graphes que les méthodes globales, ou encore divisives. La propagation de labels est une méthode locale, qui a l'avantage d'être rapide et applicable à des graphes de plusieurs millions de nœuds et d'arêtes mais elle présente certains inconvénients, à savoir de mauvaises propagations qui peuvent donner des communautés géantes, une forte instabilité due au non déterminisme de l'algorithme et l'impossibilité de trouver des communautés chevauchantes. C'est en ce sens que notre première contribution fut *i*) de proposer une version améliorée de la propagation de labels en y incluant une stabilisation par recherche de cœurs et la mise en place de barrages artificiels pour éviter de mauvaises propagations. La seconde contribution *ii*) fut d'améliorer la méthode précédente pour le chevauchement en y incluant une fonction d'appartenance permettant de détecter des nœuds pouvant appartenir à plusieurs communautés. Plusieurs fonctions d'appartenance fondées sur la densité et le cœfficient de clustering sont proposées en vue d'une étude comparative. Enfin, nous avons proposé *iii*) une implémentation MapReduce de notre propagation de labels pour travailler sur de plus grands graphes ayant au moins plusieurs millions de nœuds et d'arêtes. La propagation de labels dans sa forme asynchrone présenta une difficulté pour le parallélisme à laquelle nous

avons répondu.

5.1 Contributions algorithmiques

Pour répondre à la problématique de détection de communautés disjointes, et remédier aux problèmes de la propagations de labels, nous avons proposé la propagation de labels avec barrages qui a montré des résultats encourageants en termes de qualité de partionnement sur des graphes sociaux.

Nous avons proposé deux versions fondées sur la détection de cœurs et sur des matrices de fréquence. La première méthode était fondée sur la création de plusieurs matrices de fréquence alimentées avec différents niveaux de barrages. La seconde méthode consiste à alimenter une seule matrice de fréquence mais avec différents niveaux de barrages. La première méthode nécessite l'intervention d'une mesure de qualité pour savoir quelle matrice serait capable de donner le meilleur partitionnement. La seconde méthode, notée PLBS, nécessite de donner un intervalle sur lequel les propagations de labels avec différents niveaux de barrages alimenteraient la matrice de fréquence. Les résultats pour la simple propagation de labels avec barrages ont montré que le fait de mettre des barrages artificiels pouvait améliorer la qualité de partitionnement comme sur le réseau footballistique. Cependant, il y certains cas où la mise en place de barrages semble inutile, comme le cas du réseau de collaboration scientifique où les communautés sont déjà bien définies. Concernant les méthodes de détection par cœurs, PLBS montre des résultats très satisfaisants, notamment en alimentant la la matrice de co-fréquence de 0 à 30 % de barrages. En alimentant une matrice de co-fréquence par différents niveaux de labels, le système assure que les nœuds avec une forte probabilité d'être ensemble auront une valeur élevée au sein de la matrice. Si le nombre de barrages est trop grand, le risque est d'obtenir dans le pire des cas un nœud correspondant à une communauté, ce qui est équivalent à ce que la diagonale de la matrice de co-occurence ne soit pas vide. Cependant, la diagonale n'est pas prise en compte pour la création des composantes connexes (ce qui est un avantage de la solution). On souhaiterait néanmoins trouver un intervalle pour améliorer le partitionnement avec un certain pas et ne pas détériorer la qualité de partionnement.

Nous nous sommes également focalisés sur l'ordre de visite des nœuds lors du processus par propagation de labels. Les expérimentations ont montré que l'ordre avait une incidence à la fois sur la stabilisation et sur la qualité de partitionnement.

Pour répondre à la problématique de détection de communautés chevauchantes, nous avons voulu améliorer notre algorithme basique de détection de cœurs par propagation de labels en utilisant à la fois l'information topologique sur les structures communautaires et l'information sur les arêtes, notamment

celle concernant la pondération du graphe par utilisation de la matrice de fréquence. Les fonctions d'appartenances fondées sur la centralité de nœuds et sur le cœfficient de clustering ont montré des résultats satisfaisants en matière de qualité. L'une des forces des méthodes proposées est que l'algorithme permet de répliquer certains nœuds dans des communautés différentes autant de fois que nécessaire, c'est-à-dire qu'un nœud peut appartenir à une ou plusieurs communautés suivant la fonction utilisée. Mais cela nécessite, pour un nœud candidat au chevauchement, de tester toutes les combinaisons avec les communautés qui lui sont liées. Cela a pour conséquence une augmentation du temps d'exécution du programme informatique, notamment lorsqu'il y a beaucoup de communautés autour d'un nœud. Ainsi, nous avons pu observer, dans notre implémentation, que si un nœud était lié à beaucoup de communautés, la taille du vecteur comprenant toutes les combinaisons pouvait devenir gigantesque et ralentir l'exécution informatique. Nous proposons de ne pas considérer toutes les possibilités mais d'effectuer une procédure d'échantillonnage. L'objectif étant, à court terme, de pouvoir exploiter cette solution pour de grands graphes. Nous travaillons également sur d'autres mesures sociales en vue de faire une étude comparative.

Pour répondre à la problématique des grands graphes, nous avons développé une base pour la propagation de labels semi-synchrones à base de cœurs. Notre méthode est fondée sur une coloration de graphe, qui sera utilisée pour effectuer la propagation de label semi-synchrone pour la détection de cœurs. Cette méthode permet l'élaboration d'un dendrogramme. Notre modélisation Hadoop pour la détection de communautés a montré des résultats en termes de qualité de partitionnement encourageants. Cependant, le temps d'exécution reste trop important. Nos observations ont montré que la coloration sur les nœuds du graphe ne suivait pas une loi uniforme. Un nombre réduit de couleurs couvre la majeure partie du réseau alors que la majorité des couleurs ne couvre qu'une infime partie du réseau. Cela a pour conséquence que notre méthode prendra la majeure partie du temps à la mise à jour de labels d'un faible nombre de nœuds. Pour remédier à ce problème, nous avons déjà fusionné des couleurs de telle sorte qu'il n'existe pas de connexions entre les nœuds de ces couleurs, ce qui a pu réduire le temps d'exécution. Une piste, pour notre modèle Hadoop, serait d'analyser la fréquence de mise à jour des nœuds des labels et de ne pas effectuer de mise à jour de certains nœuds. Par exemple, un nœud connecté à une communauté dont le label ne change plus depuis un certain nombre d'itérations pourrait ne plus voir son label modifié. Nous développons actuellement une solution in-memory en utilisant Apache Spark.

5.2 Perspectives

Suite à l'analyse des résultats des expérimentations portant sur la propagation de labels avec barrages, les questions en suspens portent sur :

— le nombre de barrages nécessaires au PLAB pour obtenir la meilleure solution.
— l'intervalle à considérer pour le PBLS pour obtenir des résultats satisfaisants.

En considérant des cliques connectées, il paraît nécessaire de mettre en place des barrages entre elles. Cependant, mettre davantage de barrages ne semble pas apporter une amélioration significative dans le partitionnement de graphe, cela aurait plutôt tendance à détériorer la qualité de partitionnement.

Une des pistes que nous étudions concerne l'information de certaines régions du graphe. Nous pensons interdire la mise en place de barrages dans des zones denses ayant un fort taux de triangles. Cela pourrait s'effectuer en observant les voisinages des arêtes et en étudiant la structure topologique des communautés itération après itération en termes de pourcentages de barrages. Nos recherches portent également sur l'ordre de la mise à jour des labels . En utilisant l'information fondée sur le nœud et son horizon (à quelques arêtes), nous souhaitons créer un ordre de visite de mise à jour des labels des nœuds évitant de mauvaises propagations et permettant une convergence plus rapide de l'algorithme.

Nos méthodes de détection de communautés chevauchantes nous incitent à chercher à diminuer le temps d'exécution, tout en les appliquant à de plus grands graphes grâce à un modèle parallèle et distribué. Le nombre important de mesures sociales nous permettra également de faire une étude comparative portant sur le taux de chevauchement entre communautés.

Notre modèle Hadoop permet de traiter des graphes de plusieurs millions de nœuds et d'arêtes. Cependant, la coloration, qui est la base de cet algorithme, présente des inconvénients surtout lorsqu'un très faible nombre de nœuds a la même couleur. Cela nécessite de faire une mise à jour du graphe dans son ensemble, alors qu'un très faible nombre de nœuds est concerné. Une solution in-memory serait appréciable pour faire une étude comparative portant sur les volumétries traîtées et le temps d'exécution avec le modèle Hadoop.

En observant les communautés d'Amazon et en allant sur le site de vente en ligne, nous nous sommes rendu compte d'une grande similarité entre les éléments des communautés et les recommandations. Nous pensons que notre modèle pourrait être la base pour un système de recommandation à grande échelle. Cela constitue une piste de recherche.

Chapitre 6

Glossaire

Nous proposons un glossaire permettant de définir certaines notions, principalement liées au graphe. Dans ce cadre, nous considérons un graphe $G = (V, E)$ avec V un ensemble de sommets et E un ensemble d'arêtes E. $|V|$ dénote le nombre de sommets du graphe et $|E|$ le nombre d'arêtes.

6.1 Définitions relatives aux graphes

Benchmark LFR : Le benchmark LFR (Lancichinetti-Fortunato-Radicchi) est un algorithme de création de graphes aléatoires dont les caractéristiques se veulent aussi proches que des réseaux complexes. Les avantages de l'algorithme sur les autres méthodes est qu'il tient compte de l'hétérogénéité de la distribution des degrés des noeuds et des tailles de communautés. L'algorithme demande un certains nombre de paramètres :

- n : le nombre de noeuds
- k : le degré interne moyen des communautés
- maxk : le degré maximum
- mu : l paramètre de mixage
- t_1 : exposant pour la séquence de degré
- t_2 : exposant pour la distribution des tailles des communautés
- minc : taille minimum des communautés
- maxc : taille maximum des communautés
- o_n : nombre de noeuds chevauchants
- o_m : nombre de communautés auxquelles appartiennent les noeuds chevauchants

La bordure (ou frontière) d'une communauté est l'ensemble des nœuds ayant au moins un lien vers une autre communauté.

Coloriage : Il s'agit d'attribuer une couleur à chacun des sommets d'un graphe de manière que deux sommets reliés par une arête soient de couleur différente.

Graphe complet : Un graphe complet est un graphe où chaque sommet est relié à tous les autres. On note K_n le graphe complet à n sommets où chaque sommet est de degré $n-1$.

Graphe nul : Si l'on considère un graphe G, le modèle nul (ou graphe nul) consiste en la création d'un graphe G' respectant la distribution des degrés des noeuds avec des arêtes posées de manière aléatoire.

Clique : sous-ensemble de sommets d'un graphe dont le sous-graphe induit est complet, c'est-à-dire que deux sommets quelconques de la clique sont toujours adjacents. Une **K-clique** est un sous-ensemble de k sommets tous adjacents (sous-graphe complet), et deux k-cliques sont adjacentes si elles partagent $k-1$ sommets.

Clique maximum : Une clique maximum d'un graphe est une clique dont le cardinal est le plus grand (c'est-à-dire qu'elle possède le plus grand nombre de sommets).

Contraction : Une contraction consiste à supprimer une arête d'un graphe en fusionnant les deux extrémités. Ainsi, la contraction G/e d'une arête e_{xy} à un sommet x rend le sommet x adjacent à tous les voisins précédents de y. On retrouve les premiers principes de contraction avec les méthodes multi-niveau de partitionnement de graphe (Karypis et Kumar, 1995) (Karypis et Kumar, 1998a) (Karypis et Kumar, 1998b) (Hendrickson et Leland, 1995b) (Hendrickson et Leland, 1995a) qui seront utilisées par la suite pour les méthodes agglomératives comme Louvain ou celle de Rotta (Rotta et Noack, 2011) ou de Noack (Noack et Rotta, 2009).

Dendrogramme : un dendrogramme est une représentation graphique sous forme d'arbre binaire permettant l'observation d'une hiérarchie des parties (communautés) au sein du graphe. On peut ainsi observer pour un noeud sa communauté, mais également les imbrications entre communautés jusqu'à la racine.

Frontière des arêtes : Les arêtes menant d'une partie d'un graphe au reste du graphe.

Frontière intérieure des sommets : Les sommets d'une partie d'un graphe reliées au reste du graphe.

Frontière extérieure des sommets : Les sommets du reste d'un graphe reliées à une partie du graphe.

Moyenne harmonique : La moyenne harmonique H de nombres réels strictement positifs $x_1, ..., x_n$ est définie comme étant $H = \frac{n}{\frac{1}{x_1}+...+\frac{1}{x_n}}$ La moyenne harmonique est utilisée lorsqu'on veut déterminer un rapport moyen, dans un domaine où il existe des liens de proportionnalité inverse.

Bibliographie

Balázs Adamcsek, Gergely Palla, Illés J Farkas, Imre Derényi et Tamás Vicsek : Cfinder : locating cliques and overlapping modules in biological networks. *Bioinformatics*, 22(8):1021–1023, 2006.

Réka Albert, Hawoong Jeong et Albert-László Barabási : Internet : Diameter of the world-wide web. *Nature*, 401(6749):130–131, 1999.

LNF Ana et Anil K Jain : Robust data clustering. *In Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–128. IEEE, 2003.

George E Andrews : The theory of partitions, volume 2 of encyclopedia of mathematics and its applications, 1976.

Ching Avery : Giraph : Large-scale graph processing infrastructure on hadoop. *Proceedings of the Hadoop Summit. Santa Clara*, 11, 2011.

David A Bader, Shiva Kintali, Kamesh Madduri et Milena Mihail : Approximating betweenness centrality. *In Algorithms and Models for the Web-Graph*, pages 124–137. Springer, 2007.

Bahman Bahmani, Ravi Kumar et Sergei Vassilvitskii : Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 2012.

Albert-László Barabási et Réka Albert : Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

Albert-László Barabási et Eric Bonabeau : Scale-free networks. *Scientific American*, 288(5):50–59, 2003.

Stephen T Barnard : Pmrsb : Parallel multilevel recursive spectral bisection. *In Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, page 27. ACM, 1995.

Stephen T Barnard et Horst D Simon : Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency : Practice and experience*, 6(2):101–117, 1994.

ER Barnes et AJ Hoffman : On bounds for eigenvalues of real symmetric matrices. *Linear Algebra and its Applications*, 40:217–223, 1981.

John Arundel Barnes : *Class and committees in a Norwegian island parish.* Plenum New York, 1954.

Vladimir Batagelj et Andrej Mrvar : Pajek datasets, 2006.

Claude Berge : *Les nombres fondamentaux de la théorie des graphes* , volume 361. Théorie des Graphes, éditions Dunod, 1958.

Marcelo Blatt, Shai Wiseman et Eytan Domany : Superparamagnetic clustering of data. *Physical review letters*, 76(18):3251, 1996.

Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte et Etienne Lefebvre : Fast unfolding of communities in large networks. *Journal of statistical mechanics : theory and experiment*, 2008(10):P10008, 2008.

Phillip Bonacich : Technique for analyzing overlapping memberships. *Sociological methodology*, 4:176–185, 1972.

Ulrik Brandes : A faster algorithm for betweenness centrality*. *Journal of mathematical sociology*, 25(2):163–177, 2001.

Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski et Dorothea Wagner : On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, 2008.

Ronald L Breiger : The duality of persons and groups. *Social forces*, 53 (2):181–190, 1974.

Mingming Chen, Tommy Nguyen et Boleslaw K Szymanski : On measuring the quality of a network community structure. *In Social computing (SocialCom), 2013 international conference on*, pages 122–127. IEEE, 2013.

Mingming Chen, Tommy Nguyen et Boleslaw K. Szymanski : A new metric for quality of network community structure. *CoRR*, abs/1507.04308, 2015.

Avery Ching : Scaling apache giraph to a trillion edges. *Facebook Engineering blog*, page 25, 2013.

Fan R. K. Chung : *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92).* American Mathematical Society, décembre 1996. ISBN 0821803158.

Aaron Clauset, Mark EJ Newman et Cristopher Moore : Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

Aaron Clauset, Cosma Rohilla Shalizi et Mark EJ Newman : Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009a.

Aaron Clauset, Cosma Rohilla Shalizi et Mark EJ Newman : Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009b.

Linda M Collins et Clyde W Dent : Omega : A general formulation of the rand index of cluster recovery suitable for non-disjoint solutions. *Multivariate Behavioral Research*, 23(2):231–242, 1988.

Gennaro Cordasco et Luisa Gargano : Label propagation algorithm : a semi-synchronous approach. *International Journal of Social Network Mining*, 1(1):3–26, 2012.

Qiguo Dai, Maozu Guo, Yang Liu, Xiaoyan Liu et Ling Chen : Mlpa : Detecting overlapping communities by multi-label propagation approach. *In Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 681–688. IEEE, 2013.

Allison Davis, Burleigh Bradford Gardner et Mary R Gardner : *Deep South : A social anthropological study of caste and class*. Univ of South Carolina Press, 2009.

Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara et Alessandro Provetti : Enhancing community detection using a network weighting strategy. *Information Sciences*, 222:648–668, 2013.

Jeffrey Dean et Sanjay Ghemawat : Mapreduce : simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

L. Donetti et M. A. Muñoz : Detecting network communities : a new systematic and efficient algorithm. *Journal of Statistical Mechanics : Theory and Experiment*, 10:12, octobre 2004.

Jordi Duch et Alex Arenas : Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005.

Lei Fang, Qun Yang, Jiawen Wang et Weihua Lei : Signed network label propagation algorithm with structural balance degree for community detection. *In International Conference on Smart Homes and Health Telematics*, pages 427–435. Springer, 2016.

Miroslav Fiedler : Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

Miroslav Fiedler : A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4):619–633, 1975.

Santo Fortunato : Community detection in graphs. *Physics Reports*, 486 (3):75–174, 2010.

Linton C Freeman : Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1978.

Yaotian Fu et Philip W Anderson : Application of statistical mechanics to np-complete problems in combinatorial optimisation. *Journal of Physics A : Mathematical and General*, 19(9):1605, 1986.

Nishant M Gandhi et Rajiv Misra : Performance comparison of parallel graph coloring algorithms on bsp model using hadoop. *In Computing, Networking and Communications (ICNC), 2015 International Conference on*, pages 110–116. IEEE, 2015.

Robert Geisberger, Peter Sanders et Dominik Schultes : Better approximation of betweenness centrality. *In Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pages 90–100. Society for Industrial and Applied Mathematics, 2008.

David Gfeller, Jean-Cédric Chappelier et Paolo De Los Rios : Finding instabilities in the community structure of complex networks. *Physical Review E*, 72(5):056135, 2005.

M. Girvan et M. E. J. Newman : Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002a.

Michelle Girvan et Mark EJ Newman : Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12): 7821–7826, 2002b.

Murray Glanzer et Robert Glaser : Techniques for the study of group structure and behavior : Ii. empirical studies of the effects of structure in small groups. *Psychological Bulletin*, 58(1):1, 1961.

Pablo M Gleiser et Leon Danon : Community structure in jazz. *Advances in complex systems*, 6(04):565–573, 2003.

Gene H Golub : Cf van loan matrix computations. *The Johns Hopkins*, 1996.

Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson et Carlos Guestrin : Powergraph : Distributed graph-parallel computation on natural graphs. *In OSDI*, volume 12, page 2, 2012.

Steve Gregory : An algorithm to find overlapping community structure in networks. *In Knowledge discovery in databases : PKDD 2007*, pages 91–102. Springer, 2007.

Steve Gregory : Finding overlapping communities in networks by label propagation. *New Journal of Physics*, 12(10):103018, 2010.

R. Guimera, M. Sales-Pardo et L.A.N. Amaral : Modularity from fluctuations in random graphs and complex networks. *Physical Review E*, 70 (2):025101, 2004.

Roger Guimera et Luis A Nunes Amaral : Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900, 2005.

Kenneth M Hall : An r-dimensional quadratic placement algorithm. *Management science*, 17(3):219–229, 1970.

Bruce Hendrickson et Robert Leland : An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995a.

Bruce Hendrickson et Robert W Leland : A multi-level algorithm for partitioning graphs. *SC*, 95:28, 1995b.

George C Homans : The human group new york. *Harpers*, 1950.

Roger A Horn et Charles R Johnson : Matrix analysis cambridge university press. *New York*, 1985.

Xuegang Hu, Wei He, Huizong Li et Jianhan Pan : Role-based label propagation algorithm for community detection. *CoRR*, abs/1601.06307, 2016.

Lawrence Hubert et Phipps Arabie : Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

Panos Kalnis, Karim Awara, Hani Jamjoom et Zuhair Khayyat : Mizan : Optimizing graph mining in large parallel systems. Rapport technique, King Abdullah University of Science and Technology, 2012.

Rushed Kanawati : Licod : Leaders identification for community detection in complex networks. *In Privacy, Security, Risk and Trust (PASSAT) and IEEE Third Inernational Conference on Social Computing (SocialCom)*, pages 577–582. IEEE, 2011.

U Kang, Brendan Meeder, Evangelos E Papalexakis et Christos Faloutsos : Heigen : Spectral analysis for billion-scale graphs. *IEEE Transactions on knowledge and data engineering*, 26(2):350–362, 2014.

U Kang, Charalampos E Tsourakakis et C.Faloutsos : Pegasus : A peta-scale graph mining system implementation and observations. *In Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pages 229–238. IEEE, 2009.

Ravi Kannan, Santosh Vempala et Adrian Vetta : On clusterings : Good, bad and spectral. *Journal of the ACM (JACM)*, 51(3):497–515, 2004.

Brian Karrer, Elizaveta Levina et Mark EJ Newman : Robustness of community structure in networks. *Physical Review E*, 77(4):046119, 2008.

George Karypis et Vipin Kumar : Multilevel graph partitioning schemes. *In ICPP (3)*, pages 113–122, 1995.

George Karypis et Vipin Kumar : A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20 (1):359–392, 1998a.

George Karypis et Vipin Kumar : Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1):96–129, 1998b.

Stephen Kelley : *The existence and discovery of overlapping communities in large-scale networks*. Thèse de doctorat, RENSSELAER POLYTECHNIC INSTITUTE, 2009.

Anne-Marie Kermarrec, Erwan Le Merrer, Bruno Sericola et Gilles Trédan : Second order centrality : Distributed assessment of nodes criticity in complex networks. *Computer Communications*, 34(5):619–628, 2011.

Zuhair Khayyat, Karim Awara, Amani Alonazi, Hani Jamjoom, Dan Williams et Panos Kalnis : Mizan : a system for dynamic load balancing in large-scale graph processing. *In Proceedings of the 8th ACM European Conference on Computer Systems*, pages 169–182. ACM, 2013.

Yehuda Koren, Liran Carmel et David Harel : Ace : A fast multiscale eigenvectors computation for drawing huge graphs. *In Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pages 137–144. IEEE, 2002.

Valdis Krebs : Books about us politics. *unpublished, http ://www. orgnet. com*, 2004.

S Kullback : Statistics and information theory. *J. Wiley and Sons, New York*, 1959.

Konstantin Kuzmin, Mingming Chen et Boleslaw K Szymanski : Parallelizing slpa for scalable overlapping community detection. *Scientific Programming*, 2015:4, 2015.

Andrea Lancichinetti, Santo Fortunato et János Kertész : Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, 2009.

Andrea Lancichinetti, Santo Fortunato et Filippo Radicchi : Benchmark graphs for testing community detection algorithms. *Physical review E*, 78 (4):046110, 2008.

Andrea Lancichinetti, Filippo Radicchi, José J Ramasco et Santo Fortunato : Finding statistically significant communities in networks. *PloS one*, 6(4):e18961, 2011.

Cornelius Lanczos : *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

Conrad Lee, Fergal Reid, Aaron McDaid et Neil Hurley : Detecting highly overlapping community structure by greedy clique expansion. *In SNAKDD Workshop*, page 4533–42, 2010.

Ian XY Leung, Pan Hui, Pietro Lio et Jon Crowcroft : Towards real-time community detection in large networks. *Physical Review E*, 79(6):066107, 2009.

Wei Liu, Xingpeng Jiang, Matteo Pellegrini et Xiaofan Wang : Discovering communities in complex networks by edge label propagation. *Scientific reports*, 6, 2016.

Hao Lou, Shenghong Li et Yuxin Zhao : Detecting community structure using label propagation with weighted coherent neighborhood propinquity. *Physica A : Statistical Mechanics and its Applications*, 392(14):3095–3105, 2013.

László Lovász : *Combinatorial problems and exercises*, volume 361. American Mathematical Soc., 1993.

Yucheng Low, Joseph E. Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin et Joseph M. Hellerstein : Graphlab : A new framework for parallel machine learning. *CoRR*, abs/1408.2041, 2014.

Robert Harry Lowie : Social organization. 1950.

R Duncan Luce et Albert D Perry : A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.

David Lusseau, Karsten Schneider, Oliver J Boisseau, Patti Haase, Elisabeth Slooten et Steve M Dawson : The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser et Grzegorz Czajkowski : Pregel : a system for large-scale graph processing. *In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

Spiros Mancoridis, Brian S Mitchell, Chris Rorres, Yih-Farn Chen et Emden R Gansner : Using automatic clustering to produce high-level system organizations of source code. *In IWPC*, volume 98, pages 45–52, 1998.

Christopher D Manning, Prabhakar Raghavan et Hinrich Schütze : Flat clustering. *Introduction to information retrieval*, pages 350–374, 2008.

Claire P Massen et Jonathan PK Doye : Thermodynamics of community structure. *arXiv preprint cond-mat/0610077*, 2006.

Aaron F McDaid, Derek Greene et Neil Hurley : Normalized mutual information to evaluate overlapping community finding algorithms. *arXiv preprint arXiv :1110.2515*, 2011.

Guy Melancon : Just how dense are dense graphs in the real world ? : a methodological note. *In Proceedings of the 2006 AVI workshop on BEyond time and errors : novel evaluation methods for information visualization*, pages 1–7. ACM, 2006.

Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara et Alessandro Provetti : Generalized louvain method for community detection in large networks. *In Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 88–93. IEEE, 2011.

Seunghyeon Moon, Jae-Gil Lee, Minseo Kang *et al.* : Scalable community detection from networks by computing edge betweenness on mapreduce. *In 2014 International Conference on Big Data and Smart Computing (BIGCOMP)*, pages 145–148. IEEE, 2014.

Jacob L Moreno : *Who shall survive*, volume 58. JSTOR, 1934.

Jacob Levy Moreno : Sociometry, experimental method and the science of society. 1951.

George Peter Murdock : Social structure. 1949.

S Nadel : The theory of social structure (cohen and west, london). 1957.

Tamás Nepusz, Andrea Petróczi, László Négyessy et Fülöp Bazsó : Fuzzy communities and the concept of bridgeness in complex networks. *Physical Review E*, 77(1):016107, 2008.

Mark EJ Newman : The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

Mark EJ Newman : Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.

Mark EJ Newman : Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

Mark EJ Newman : Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

Mark EJ Newman : Spectral methods for community detection and graph partitioning. *Physical Review E*, 88(4):042822, 2013.

Mark EJ Newman et Michelle Girvan : Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

Andrew Y. Ng, Michael I. Jordan et Yair Weiss : On spectral clustering : Analysis and an algorithm. *In ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 849–856. MIT Press, 2001.

Vincenzo Nicosia, Giuseppe Mangioni, Vincenza Carchiolo et Michele Malgeri : Extending the definition of modularity to directed graphs with overlapping communities. *Journal of Statistical Mechanics : Theory and Experiment*, 2009(03):P03024, 2009.

Andreas Noack et Randolf Rotta : Multi-level algorithms for modularity clustering. *In International Symposium on Experimental Algorithms*, pages 257–268. Springer, 2009.

Michael Ovelgonne : Distributed community detection in web-scale networks. *In Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, pages 66–73. IEEE, 2013.

Gergely Palla, Imre Derényi, Illés Farkas et Tamás Vicsek : Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.

Symeon Papadopoulos, Yiannis Kompatsiaris, Athena Vakali et Ploutarchos Spyridonos : Community detection in social media. *Data Mining and Knowledge Discovery*, 24(3):515–554, 2012.

Hao Peng, Dandan Zhao, Lin Li, Jianfeng Lu, Jianmin Han et Songyang Wu : An improved label propagation algorithm using average node energy in complex networks. *Physica A : Statistical Mechanics and its Applications*, 460:98–104, 2016.

Anthony N Pettitt et Michael A Stephens : The kolmogorov-smirnov goodness-of-fit statistic with discrete and grouped data. *Technometrics*, 19 (2):205–210, 1977.

Michel Plantié et Michel Crampes : Survey on social community detection. *In Social media retrieval*, pages 65–85. Springer, 2013.

George Pólya et Gabor Szegö : *Problems and Theorems in Analysis II : Theory of Functions. Zeros. Polynomials. Determinants. Number Theory. Geometry.* Springer Science & Business Media, 1997.

Pascal Pons et Matthieu Latapy : Computing communities in large networks using random walks. *Journal of Graph Algorithms and Applications*, 10(2): 191–218, 2006.

Alex Pothen, Horst D Simon et Kang-Pu Liou : Partitioning sparse matrices with eigenvectors of graphs. *SIAM journal on matrix analysis and applications*, 11(3):430–452, 1990.

Arnau Prat-Pérez, David Dominguez-Sal et Josep-Lluis Larriba-Pey : High quality, scalable and parallel community detection for large real graphs. *In Proceedings of the 23rd international conference on World wide web*, pages 225–236. ACM, 2014.

Ioannis Psorakis, Stephen Roberts, Mark Ebden et Ben Sheldon : Overlapping community detection using bayesian non-negative matrix factorization. *Physical Review E*, 83(6):066114, 2011.

Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto et Domenico Parisi : Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9):2658–2663, 2004.

Usha Nandini Raghavan, Réka Albert et Soundar Kumara : Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

William M Rand : Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

Jörg Reichardt et Stefan Bornholdt : Statistical mechanics of community detection. *Physical Review E*, 74(1):016110, 2006.

Aria Rezaei, Saeed Mahlouji Far et Mahdieh Soleymani : Controlled label propagation : Preventing over-propagation through gradual expansion. *arXiv preprint arXiv :1503.04694*, 2015.

E Jason Riedy, Henning Meyerhenke, David Ediger et David A Bader : Parallel community detection for massive graphs. *In International Conference on Parallel Processing and Applied Mathematics*, pages 286–296. Springer, 2011.

Jason Riedy, David A Bader et Henning Meyerhenke : Scalable multithreaded community detection in social networks. *In Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1619–1628. IEEE, 2012.

Peter Ronhovde et Zohar Nussinov : Local resolution-limit-free potts model for community detection. *Physical Review E*, 81(4):046114, 2010.

M Rosvall et CT Bergstrom : Maps of information flow reveal community structure in complex networks. *arXiv preprint physics.soc-ph/0707.0609*, 2007a.

Martin Rosvall et Carl T Bergstrom : An information-theoretic framework for resolving community structure in complex networks. *Proceedings of the National Academy of Sciences*, 104(18):7327–7331, 2007b.

Martin Rosvall et Carl T Bergstrom : Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.

Martin Rosvall et Carl T Bergstrom : Mapping change in large networks. *PloS one*, 5(1):e8694, 2010.

Randolf Rotta et Andreas Noack : Multilevel local search algorithms for modularity clustering. *Journal of Experimental Algorithmics (JEA)*, 16:2–3, 2011.

Marta Sales-Pardo, Roger Guimera, André A Moreira et Luís A Nunes Amaral : Extracting the hierarchical organization of complex systems. *Proceedings of the National Academy of Sciences*, 104(39):15224–15229, 2007.

Semih Salihoglu, Jaeho Shin, Vikesh Khanna, Ba Quan Truong et Jennifer Widom : Graft : A debugging tool for apache giraph. *In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1403–1408. ACM, 2015.

Semih Salihoglu et Jennifer Widom : Gps : a graph processing system. *In Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, page 22. ACM, 2013.

Matthew Saltz, Arnau Prat-Pérez et David Dominguez-Sal : Distributed community detection with the wcc metric. *In Proceedings of the 24th International Conference on World Wide Web*, pages 1095–1100. ACM, 2015.

Massoud Seifi, Ivan Junier, Jean-Baptiste Rouquier, Svilen Iskrov et Jean-Loup Guillaume : Stable community cores in complex networks. *In Complex Networks*, pages 87–98. Springer, 2013.

Sangwon Seo, Edward J Yoon, Jaehong Kim, Seongwook Jin, Jin-Soo Kim et Seungryoul Maeng : Hama : An efficient matrix computation with the mapreduce framework. *In Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 721–726. IEEE, 2010.

Devavrat Shah et Tauhid Zaman : Community detection in networks : The leader-follower algorithm. *In Workshop on Networks Across Disciplines : Theory and Application*, pages 1–8, 2010.

Hua-Wei Shen et Xue-Qi Cheng : Spectral methods for the detection of network community structure : a comparative analysis. *Journal of Statistical Mechanics : Theory and Experiment*, 2010(10):P10020, 2010.

Huawei Shen, Xueqi Cheng, Kai Cai et Mao-Bin Hu : Detect overlapping and hierarchical community structure in networks. *Physica A : Statistical Mechanics and its Applications*, 388(8):1706–1712, 2009.

Jianbo Shi et Jitendra Malik : Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

Lixin Shi et Junxing Zhang : Community detection using a potential-based label propagation algorithm. *Future Communication Technology (2 Volume Set)*, 51:353, 2014a.

Lixin Shi et Junxing Zhang : Community detection using robust label propagation algorithm. *Sensors & Transducers*, 163(1):198, 2014b.

Christian L Staudt et Henning Meyerhenke : Engineering high-performance community detection heuristics for massive graphs. *In Parallel Processing (ICPP), 2013 42nd International Conference on*, pages 180–189. IEEE, 2013.

Lovro Šubelj et Marko Bajec : Unfolding communities in large complex networks : Combining defensive and offensive label propagation for core extraction. *Physical Review E*, 83(3):036103, 2011.

Vincent A Traag, Gautier Krings et Paul Van Dooren : Significant scales in community structure. *Scientific reports*, 3, 2013.

Jeffrey Travers et Stanley Milgram : An experimental study of the small world problem. *Sociometry*, pages 425–443, 1969.

Serafeim Tsironis, Mauro Sozio, Michalis Vazirgiannis et LIX-Ecole Polytechnique : Accurate spectral clustering for community detection in mapreduce. *In Advances in Neural Information Processing Systems (NIPS) Workshops*. Citeseer, 2013.

Leslie G Valiant : A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

CJ van Rijsbergen : Information retrieval. 1979, 1979.

Ulrike Von Luxburg : A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

David J Wales et Jonathan PK Doye : Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116, 1997.

Jianxin Wang, Jun Ren, Min Li et Fang-Xiang Wu : Identification of hierarchical and overlapping functional modules in ppi networks. *IEEE transactions on nanobioscience*, 11(4):386–393, 2012.

Stanley Wasserman et Katherine Faust : *Social network analysis : Methods and applications*, volume 8. Cambridge university press, 1994.

Zhi-Hao Wu, You-Fang Lin, Steve Gregory, Huai-Yu Wan et Sheng-Feng Tian : Balanced multi-label propagation for overlapping community detection in social networks. *Journal of Computer Science and Technology*, 27(3):468–479, 2012.

Jierui Xie et Boleslaw K Szymanski : Labelrank : A stabilized label propagation algorithm for community detection in networks. *In Network Science Workshop (NSW), 2013 IEEE 2nd*, pages 138–143. IEEE, 2013.

Jierui Xie, Boleslaw K Szymanski et Xiaoming Liu : Slpa : Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process. *In Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, pages 344–349. IEEE, 2011.

Yan Xing, Fanrong Meng, Yong Zhou, Mu Zhu, Mengyu Shi et Guibin Sun : A node influence based label propagation algorithm for community detection in networks. *The Scientific World Journal*, 2014, 2014.

Jaewon Yang et Jure Leskovec : Structure and overlaps of communities in networks. *SNAKDD*, 2012.

Jaewon Yang et Jure Leskovec : Overlapping community detection at scale : a nonnegative matrix factorization approach. *In Proceedings of the sixth ACM international conference on Web search and data mining*, pages 587–596. ACM, 2013.

Jaewon Yang et Jure Leskovec : Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213, 2015.

W.W. Zachary : An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker et Ion Stoica : Spark : cluster computing with working sets. *In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

Aiping Zhang, Guang Ren, Yejin Lin, Baozhu Jia, Hui Cao, Jundong Zhang et Shubin Zhang : Detecting community structures in networks by label propagation with prediction of percolation transition. *The Scientific World Journal*, 2014, 2014.

Shihua Zhang, Rui-Sheng Wang et Xiang-Sun Zhang : Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A : Statistical Mechanics and its Applications*, 374(1):483–490, 2007.

Xian-Kun Zhang, Song Fei, Chen Song, Xue Tian et Yang-Yue Ao : Label propagation algorithm based on local cycles for community detection. *International Journal of Modern Physics B*, 29(05):1550029, 2015.

Yuzhou Zhang, Jianyong Wang, Yi Wang et Lizhu Zhou : Parallel community detection on large networks with propinquity dynamics. *In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 997–1006. ACM, 2009.

Kuang Zhoua, Arnaud Martin, Quan Pan et Zhun-Ga Liu : Evidential label propagation algorithm for graphs. *In Information Fusion (FUSION), 2016 19th International Conference on*, pages 1316–1323. IEEE, 2016.

Liang Zong-Wen, Li Jian-Ping, Yang Fan et Athina Petropulu : Detecting community structure using label propagation with consensus weight in complex network. *Chinese Physics B*, 23(9):098902, 2014.