HEC MONTRÉAL

Algorithmes parallèles en détection de communautés dans les réseaux complexes

par
Philippe Gagnon

Sciences de la gestion
(Analytique d'affaires)

Mémoire présenté en vue de l'obtention
du grade de maîtrise ès sciences en gestion
(M. Sc.)

Janvier 2017

Résumé

La détection de communautés dans les réseaux consiste en la recherche de regroupements de noeuds fortement connectés ensemble, tout en présentant des relations relativement éparses avec les autres noeuds externes. Le problème est $\mathcal{NP}$-complet dans la grande majorité des définitions formelles acceptées par la communauté scientifique, ce qui rend sa résolution en des temps utiles particulièrement difficile. Or, le problème est aussi particulièrement important, et a des applications dans des domaines diversifiés tels que la sociologie, l'informatique, l'ingénierie et la biologie. Une avenue permettant l'étude de réseaux de plus grande taille est la parallélisation des algorithmes. Toutefois, la majorité des algorithmes de pointe en détection de communautés consistent en des procédures inhéremment séquentielles, ce qui rend cette tâche difficile. Ce mémoire étudie des méthodes et relaxations visant à faciliter la parallélisation. Dans un premier temps, les réseaux complexes sont présentés et l'état de l'art en détection de communautés est exploré, et certaines bases en parallélisme en informatique sont ensuite expliquées. Des algorithmes parallèles sont ensuite développées, implémentées et évaluées. Ces algorithmes seront distribués au public sous une licence libre.

Mots-clefs Réseaux complexes, Détection de communautés, Algorithmes parallèles

Abstract

Community detection in networks is the process by which groups of elements that are densely connected together, but sparsely connected with other elements, are identified. Under most of its accepted definitions, the problem is $\mathcal{NP}$-complete, which makes solving it particularly difficult. It is also very important, as it has wide-ranging applications in diverse fields such as sociology, informatics, engineering and biology. A popular avenue that is often exploited to make slow algorithms perform faster is to parallelize their execution. However, in community detection, most state-of-the-art algorithms are of an inherently sequential nature, which makes parallelization non-trivial. This thesis studies methods and relaxations that aim to alleviate this problem. We first introduce network science and review the state of the art in community detection, and discuss parallelization in a general context. We then develop and implement two parallel algorithms for community detection, and benchmark their outputs against a variety of evaluation datasets. These algorithms are set to be released to the public under a free software license.

Keywords Complex Networks, Community Detection, Parallel Algorithms

Remerciements

Dans un premier temps, je tiens à remercier mes directeurs de recherche, **Gilles Caporossi** et **Sylvain Perron**. Ces derniers ont su me guider et me motiver non seulement tout au long de ce processus de rédaction de mémoire, mais aussi tout au long de mes études à HEC Montréal. Je tiens à souligner ma grande appréciation face à leur flexibilité et à la confiance qu'ils m'ont accordée. Je tiens aussi à remercier **Églantine Camby** pour ses commentaires constructifs dans le cadre de la révision de ce travail.

En second lieu, je tiens à remercier mes parents, qui me soutiennent inconditionnellement depuis des temps immémoriaux. Ces derniers connaissent mieux que quiconque la valeur que je leur accorde.

Troisièmement, je remercie mes amis et collèges étudiants aux cycles supérieurs à HEC Montréal. Ils ont, par leur présence, rendu mon passage à HEC Montréal inoubliable. Je tiens toutefois à mentionner spécialement mes amis **Morgan Martin** et **Lou Bonatesta**, sans qui ces deux dernières années n'auraient jamais été les mêmes.

Finalement, je me dois de remercier l'**Institut des sciences mathématiques** et la **Fondation HEC Montréal** pour leur soutien financier dans le cadre de mes études de deuxième cycle.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Avant-propos

Le chapitre 4 de ce mémoire est un manuscrit intitulé «Parallel methods for community detection in complex networks », écrit sous la supervision et avec la collaboration de Gilles Caporossi et Sylvain Perron. Ce manuscrit est actuellement en préparation. La définition et la caractérisation du problème, la confection des algorithmes, leur implémentation et leur évaluation sont l'oeuvre principale de Philippe Gagnon. Ce manuscrit est écrit en anglais puisqu'il s'agit de la *lingua franca* de l'informatique et de la recherche opérationnelle, domaines principaux dans lesquels se situe ce travail.

Chapitre 1

Introduction

En 1736, Euler proposait sa maintenant célèbre solution au problème des ponts de Königsberg [Eul41]. La publication de ce travail a lancé une sous-discipline mathématique nommée théorie des graphes, qui consiste en l'étude de structures mathématiques représentant des propriétés relationnelles entre différents éléments. L'intérêt pour ces structures découle de leur pouvoir descriptif. Malgré leur généralité qui découle du fait qu'ils puissent être utilisés pour représenter des relations de tous types, les graphes sont en mesure d'être extrêmement explicites par le rattachement de propriétés à leurs éléments et leurs liens. L'étude des graphes sous cet angle est une discipline scientifique connue sous le nom de science des réseaux.

Les réseaux peuvent être utilisés pour modéliser des systèmes provenant de divers horizons. En particulier, ces derniers ont été utilisés pour représenter des relations sociales, des systèmes technologiques, des systèmes biologiques et écologiques, ou encore des réseaux de connaissances. Les graphes constitués afin de modéliser des systèmes réels diffèrent généralement de graphes plus simples tels que des grilles ou des graphes aléatoires. Les premiers présentent souvent des caractéristiques topologiques non triviales, et sont par conséquent appelés réseaux complexes [New03].

La caractéristique topologique étudiée principalement dans le cadre de ce mémoire est la structure de communauté. Un réseau exhibe une structure de communauté si les éléments qui le composent peuvent être regroupés de manière à ce que les éléments faisant partie du même groupe soient densément connectés, mais avec des connexions éparses envers les éléments membres des autres groupes.

La recherche de communautés dans un réseau est un problème communément nommé détection de communauté. Ce problème est extrêmement difficile à résoudre puisqu'il est $\mathcal{NP}$-complet. Or, un grand nombre de jeux de données relationnelles est amassé sur une base régulière, et la taille de ces derniers ne cesse de croitre.

La confection d'algorithmes heuristiques ou approximatifs capables de détecter des structures de communauté de bonne qualité en temps restreints a donc fait l'objet de multiples études. Toutefois, les principaux algorithmes de pointe du domaine sont basés sur des principes itératifs inhéremment

séquentiels. Il est donc difficile de les exécuter en parallèle afin d'exploiter des ressources de calcul additionnelles, et ainsi obtenir des solutions à des problèmes de grande taille dans des temps raisonnables.

Ce mémoire vise à contribuer à la résolution de ce problème en proposant des méthodes parallèles pour détecter des communautés dans des réseaux complexes de grande taille. L'exposé est structuré ainsi : le chapitre 2 introduit et révise l'état de l'art en étude des réseaux complexes. La section 2.2 consiste en un exposé de concepts de base en théorie des graphes, et peut être outrepassée par un lecteur déjà familier avec le domaine. Le chapitre 3 consiste en une introduction au calcul parallèle, dans un contexte général. Le manuscrit «Parallel methods for community detection in complex networks » est reproduit intégralement dans le chapitre 4. Dans ce chapitre, la section 4.2 constitue une courte revue de littérature en détection de communauté, avec un accent sur les méthodes de pointe, et la sous-section 4.3.3 un exposé encore plus succinct des méthodes parallèles en détection de communauté. La brièveté de ces revues, rendue nécessaire par le format manuscrit du chapitre, rehausse la pertinence du chapitre 2, dont l'objectif est de situer le travail dans le champ plus large de la science des réseaux. Finalement, le chapitre 5 conclut le mémoire.

Chapitre 2

Les réseaux complexes

Les réseaux complexes sont des réseaux dont la caractéristique principale est de présenter des particularités topologiques que l'on ne retrouve que rarement dans des réseaux plus simples. Curieusement, l'on découvre ces particularités lorsque plusieurs systèmes que l'on retrouve dans le monde réel sont modélisés sous forme de réseau. L'objectif de ce chapitre est de fournir au lecteur un survol de la recherche de pointe en science des réseaux complexes. Le chapitre commence par une revue de travaux scientifiques traitant de modélisation de systèmes complexes sous forme de réseaux, avec la section 2.1. Ensuite, la section 2.2 présente certains concepts de base de la théorie des graphes, dont la connaissance est essentielle à la compréhension de la suite du mémoire. Finalement, la section 2.3 traite de la structure de communauté, la propriété que l'on cherche à étudier avec les algorithmes présentés au chapitre 4.

2.1 Systèmes réels sous forme de réseau

Tel que mentionné en introduction, plusieurs systèmes complexes peuvent être modélisés sous forme de réseaux et étudiés avec les outils mathématiques issus de la théorie des graphes. Plusieurs travaux précédents se sont concentrés sur l'étude de réseaux sociaux, technologiques, biologiques et d'information.

2.1.1 Réseaux sociaux

La modélisation sous forme de réseaux est fréquemment employée afin de représenter des relations sociales. Cette pratique, l'analyse de réseaux sociaux, provient des travaux de Jacob Moreno [Mor34]. Dans le cadre de l'analyse d'une problématique de fugues dans une école pour filles à New York, ce dernier a découvert que le facteur explicatif principal de la problématique était la position d'une élève dans le réseau social composé par le corps étudiant. C'est cette étude qui est créditée comme ayant lancé le champ de la sociométrie, dont l'un des principaux outils est l'analyse de réseaux sociaux [BMBL09].

Peu de temps après, Davis et al. [DGG41] étudient les relations sociales entre les femmes d'une ville du sud des États-Unis en 1936. Ces derniers sont en mesure de caractériser les relations sociales de ces femmes en fonction de leur position hiérarchique au sein du groupe. Plus tard, plusieurs autres études se sont concentrées sur l'étude des relations amicales entre différents individus [FS64, RH61].

En 1969, Milgram et Travers déterminent que deux individus aléatoires sont séparés par, en moyenne, 5.2 intermédiaires [Mil67, TM69]. Ces travaux ont donnés naissance au phénomène de «petit monde » (small-world), bien connu dans la culture populaire. C'est ce qui aura inspiré le modèle de réseau complexe «small-world ».

Plus récemment, les chercheurs se sont particulièrement intéressés aux relations d'affaires [GM78, Mar75] et aux relations sexuelles entre individus [HK07, BMS04, DSW+04]. Toutefois, un développement récent particulièrement important consiste en l'acquisition de données sur les relations sociales avec des moyens autres que l'observation directe.

Ce développement aura permis d'obtenir des données sur des réseaux sociaux d'une taille beaucoup plus importante que par le passé. En particulier, [ML12] introduit des jeux de données basés sur Facebook, Twitter and Google+, et [YL15] introduit 230 jeux de données basés sur des réseaux sociaux en ligne. Certains chercheurs ont eu accès à des réseaux composés de métadonnées concernant des communications sans-fils [BDK15], ce qui permet entre autres d'étudier la mobilité et la distribution géographiques des individus.

2.1.2 Réseaux technologiques

On regroupe généralement sous la bannière des réseaux technologiques les réseaux de distribution d'information ou de distribution électrique, qui sont couramment étudiés. Un réseau similaire fréquemment étudié est le réseau formé par les routeurs qui permettent la distribution de données à l'échelle de l'internet [BC01]. Ces réseaux sont généralement obtenus à l'aide d'observations des routes empruntées par les données pour se rendre d'un ordinateur à l'autre.

La catégorie des réseaux technologiques ne comporte toutefois pas seulement des réseaux informatiques. Certains chercheurs ont étudié les propriétés des réseaux de distribution électrique [WS98]. D'autres ont étudié, par exemple, les réseaux routiers [LGH06], ou les réseaux de trafic aérien [LHY+11].

2.1.3 Réseaux biologiques et écologiques

Un grand nombre de systèmes biologiques peuvent être représentés sous forme de réseaux. Par exemple, les réactions qui résultent de l'interaction entre différentes protéines peuvent être modélisées sous la forme d'arcs rejoignant des noeuds (représentant des protéines) [JMBO01, TLT07].

Les écologistes étudient fréquemment les interactions entre les prédateurs et leurs proies dans un environnement naturel. Ces interactions peuvent être représentées sous forme de réseaux, avec les prédateurs et les proies représentés par les noeuds, et les relations entre des derniers par les arcs

[DM04]. L'étude de ces réseaux peut, par exemple, permettre de mieux comprendre les implications de la disparition d'une espèce animale sur la chaine alimentaire [SLJRT10].

Finalement, un autre type de travaux d'intérêt consiste en études qui modélisent des réseaux de neurones [WSTB86]. Ces réseaux sont particulièrement difficiles à reconstituer, mais l'étude de ces derniers permet d'obtenir un aperçu intéressant de la structure qui sous-tend le fonctionnement du système nerveux [CS07].

2.1.4 Réseaux de connaissances

Les chercheurs se sont rapidement intéressés à l'étude des réseaux formés par les citations entre les articles scientifiques [Seg92, New01]. Ces réseaux sont par définition orientés et généralement acycliques. Les travaux dans ce domaine sont historiquement significatifs, puisque l'étude de ces réseaux qui aura pour la première fois permis l'identification de réseaux dont la distribution du degré des noeuds suit une loi de puissance [Pri65]. Il s'agit d'une caractéristique commune à des réseaux représentant plusieurs phénomènes naturels complexes (nommés réseaux invariants d'échelle).

Le réseau le plus notoire dans cette catégorie constitue sans aucun doute le Web (WWW). Ce dernier a été étudié par Barabásil et Albert [BA99], qui ont montré non seulement que la distribution des degrés des noeuds dans le réseau suit une loi de puissance, mais aussi que cette structure émerge du processus de construction du réseau. En effet, il semblerait que les liens entre les éléments s'ajoutent selon un processus nommé *attachement préférentiel*, qui dicte que les éléments ont tendance à se lier avec une plus grande probabilité avec ceux qui possèdent un plus grand nombre de liens préexistants.

Les auteurs proposent de plus que ce processus puisse sous-tendre la formation d'autres réseaux naturels. Ce constat est particulièrement significatif. En effet, si la distribution du degré des noeuds d'un réseau suit une loi de puissance, il est possible de poser une hypothèse concernant le processus qui sous-tend le phénomène de construction du réseau en tant que tel.

2.2 Concepts de base en théorie des graphes

Cette section vise à présenter certains concepts basiques de la théorie de graphes, la discipline mathématique sur laquelle repose l'étude des réseaux. L'accent dans cette section est porté sur les concepts qui sont fréquemment utilisés dans le cadre de l'étude de réseaux complexes présentant une structure de communauté. Pour un traitement plus complet, le lecteur peut se référer à [Tru94], un texte introductif très accessible.

2.2.1 Définitions

Définition 2.2.1. Un **graphe** est une paire $G = (V, E)$, avec V l'ensemble des sommets, ou noeuds, qui composent le réseau, et $E \subseteq V^2$ l'ensemble des arêtes, ou arcs, qui relient ces sommets ensemble. Deux sommets sont dits adjacents s'il existe un arc qui les relie. Le nombre de sommets présents

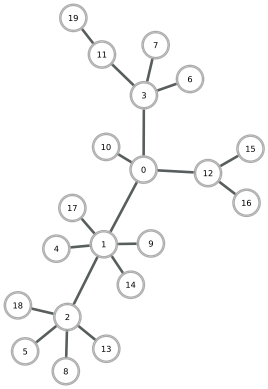

FIG. 2.2.1: Graphe non-orienté avec 20 sommets et 19 arcs. Ce graphe ne contient aucun cycle et appartient donc à la famille des arbres.

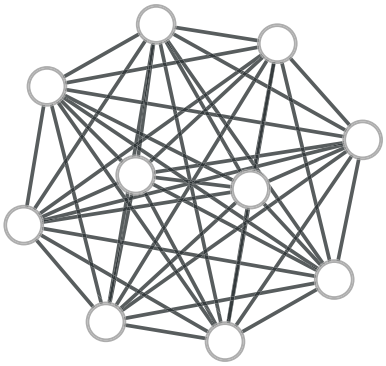

FIG. 2.2.2: Graphe complet à dix sommets

dans un graphe, dénoté $|V|$, est appelé l'ordre du graphe, tandis que le nombre d'arcs, dénoté $|E|$, est nommé la taille du graphe.

Définition 2.2.2. Un **sous-graphe** est un graphe contenu dans un autre graphe, soit $G' \subset G$.

Définition 2.2.3. Un **graphe connexe** est un graphe au sein duquel il existe un chemin entre toutes les paires de sommets.

Définition 2.2.4. Une **composante connexe** est un sous-graphe de G étant à la fois maximal et connexe.

Définition 2.2.5. Un graphe **complet** est un graphe au sein duquel il existe un arc entre toutes les paires de sommets, soit $\exists\,(u, v) \in V^2\,\forall\,u, v \in V$. Par définition, un graphe complet est connexe. Le graphe représenté dans la figure 2.2.2 est un exemple de ce type de graphe.

Définition 2.2.6. Un graphe est **orienté** s'il existe une relation directionnelle sur les arcs qui relient les sommets. Dans ce cas, E est un ensemble de paires ordonnées.

Définition 2.2.7. Un graphe est **pondéré** s'il existe une fonction de pondération $w(v \in V)$ ou $w(e \in E)$ sur ses sommets ou ses arcs, respectivement.

Définition 2.2.8. Une **boucle** est un arc $e \in E := (v, v)$, soit un arc reliant un sommet à lui-même.

Définition 2.2.9. Des arcs sont dits **multiples** s'ils relient la même paire de sommets, soit $e, e' := (u, v)$.

Définition 2.2.10. La **distance** $d(u, v)$ entre deux sommets est, dans un graphe non pondéré, le nombre d'arcs du plus court chemin entre u et v. Dans un graphe pondéré, il s'agit de la somme des poids du chemin le moins lourds entre u et v.

Définition 2.2.11. Le **degré** d'un sommet v est le nombre d'arcs qui lui sont adjacents.

Définition 2.2.12. Un **invariant** est une fonction f qui associe une valeur généralement scalaire à un graphe.

2.2.2 Indices et coefficients importants

Définition 2.2.13. Un graphe a un **degré maximal** $\Delta := \max_{v \in V} k_v$.

Définition 2.2.14. Un graphe a un **degré minimal** $\delta := \min_{v \in V} k_v$.

Définition 2.2.15. Un graphe possède une **séquence des degrés**, qui consiste en la séquence monotone non croissante dont les éléments sont les degrés des sommets du graphe.

Définition 2.2.16. La **densité** est le nombre d'arcs présents dans le réseau, comparativement avec le nombre d'arc qui pourraient possiblement exister dans un réseau de la même taille. Pour un graphe non orienté, la densité est $\frac{2|E|}{|V|\times|V-1|}$, et $\frac{|E|}{|V|\times|V-1|}$ pour un graphe orienté.

Définition 2.2.17. L'**excentricité** d'un sommet v est la distance maximum entre u et tout autre sommet v du graphe, soit $\epsilon(v) = \max_u d(v, u)$.

Définition 2.2.18. Le **diamètre** d'un graphe est l'excentricité maximum présente dans le graphe, soit $D(G) = \max_{v \in G} \epsilon(v)$.

Définition 2.2.19. Le **rayon** d'un graphe est la l'excentricité minimum présente dans le graphe, soit $r(G) = \min_{v \in G} \epsilon(v)$.

2.2.3 Structures de données

Les graphes sont généralement étudiés à l'aide d'ordinateurs. Il est donc pertinent d'étudier les structures de données utilisées pour les représenter sous forme informatique, puisque la forme qu'elles prennent ont une incidence directe sur la performance des algorithmes utilisés pour les analyser. Il existe trois structures principales pour représenter un graphe, soit la matrice d'adjacence, la liste d'adjacence et la liste d'arêtes.

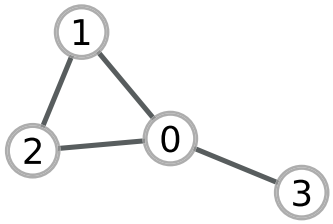

FIG. 2.2.3: Graphe simple à quatre sommets

2.2.3.1 Matrice d'adjacence

Une matrice d'adjacence est une matrice $A \coloneqq \{a_{ij}\}$ avec

$$a_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & (i,j) \notin E \end{cases}$$

pour un graphe non pondéré, et $a_{ij} = w[(i,j)]$ si le graphe est pondéré. Ainsi, la représentation sous forme de matrice d'adjacence de la figure 2.2.3 serait :

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \tag{2.2.1}$$

La matrice d'adjacence est une représentation très utile d'un point de vue mathématique, puisqu'elle permet de facilement utiliser les outils et les algorithmes issus de l'algèbre linéaire pour étudier les graphes. Toutefois, cette dernière nécéssite $\mathcal{O}(n^2)$ d'espace de stockage et de mémoire.

En outre, un avantage majeur de la matrice d'adjacence est vérifier si un arc existe entre deux sommets, ou obtenir le poids d'un arc, est une opération de complexité $\mathcal{O}(1)$ sous cette représentation.

2.2.3.2 Listes d'adjacence

La représentation sous forme de listes d'adjacence est utile lorsque l'on traite avec un graphe éparse, c'est à dire un graphe dont la taille $|E| \ll V^2$. En effet, alors que la représentation sous forme de matrice d'adjacence nécéssite $\mathcal{O}(n^2)$ d'espace de stockage et de mémoire, la structure de listes d'adjacence en nécéssite $\mathcal{O}(n)$.

La représentation sous forme de liste d'adjacence peut être conceptualisée sous la forme d'un

vecteur $\mathbf{v} \coloneqq \{\mathbf{a_{i \in V}}\}$, tel que les éléments de $\mathbf{a_i}$ soient les sommets adjacents au sommet i. Pour la figure 2.2.3, l'on aurait :

$$\begin{pmatrix} \{1,2,3\} \\ \{0,2\} \\ \{0,1\} \\ \{0\} \end{pmatrix} \tag{2.2.2}$$

À l'instar de la représentation sous forme de matrice d'adjacence, vérifier si un arc existe entre deux sommets, ou obtenir le poid d'un arc, est une opération de complexité $\mathcal{O}(n)$ dans le cadre de la représentation sous forme de listes d'adjacence. Obtenir tous les sommets adjacents à un sommet particulier est toutefois une opération plus simple et plus rapide sous ce régime, puisqu'il suffit alors de lire le vecteur $\mathbf{a_i}$ dont la taille se limite au nombre de sommets adjacents, alors que sous le régime de la matrice d'adjacence, la lecture d'une ligne entière de la matrice est requise, ce qui constitue une opération $\mathcal{O}(|V|)$.

2.2.3.3 Liste d'arêtes

La liste d'arêtes est une représentation de graphes qui prend très peu d'espace, puisqu'elle est constituée d'un vecteur $\mathbf{e} \coloneqq \{e \in E\}$. La liste d'arêtes prend donc $\mathcal{O}(|E|)$ d'espace de stockage et de mémoire. La représentation sous cette forme de la figure 2.2.3 serait :

$$\{(0,1),(0,2),(0,3),(1,2)\} \tag{2.2.3}$$

Il est toutefois difficile de travailler avec une liste d'arêtes, puisque la structure de cette dernière rend l'accès aux données moins flexible. Cette représentation est toutefois couramment utilisée afin de stocker un graphe dans un fichier pour une utilisation subséquente.

2.3 La structure de communauté

2.3.1 Définitions

2.3.1.1 Définition classique

De manière intuitive, un réseau comporte une structure de communauté s'il est possible de concevoir un regroupement de ses sommets, de telle manière que les sommets membres d'un même groupe soient fortement connectés, tout en étant faiblement connectés avec les sommets membres d'autres groupes. Cette caractérisation générale nous permet de souligner le concept sans trop le circonscrire, et permet d'introduire deux définitions traditionnelles plus formelles. Radicchi et al. [RCC+04] proposent le concept de *communauté au sens fort*, soit une communauté au sein de laquelle, pour chaque noeud, le degré interne soit plus grand que le degré externe, et le concept de *communauté au sens*

faible, au sein de laquelle le degré interne de la communauté prise globalement soit plus grand que son degré externe.

2.3.1.2 Définition moderne

Les interprétations modernes du concept de communauté diffèrent toutefois sensiblement de la définition énoncée ci-haut. En effet, Fortunato et Hric [FH16] argumentent que de nos jours, une communauté est plutôt conceptualisée sur une base statistique ; c'est-à-dire que, au lieu d'identifier une communauté à l'aide du nombre absolu de liens entre ses différents membres et les autres sommets d'un réseau, on l'identifie à l'aide de la probabilité qu'un lien soit présent entre deux sommets. Cette définition est beaucoup plus opérationnelle et aura permis la confection d'une grande variété d'algorithmes de détection de communauté, tel qu'exposé dans la section 2.3.2.

2.3.1.3 Variantes

En plus de la variante classique de structure de communauté, certaines extensions et généralisations du modèle ont été introduites afin de modéliser certaines situations que l'on retrouve dans le monde réel.

Structure de communauté classique La définition classique de structure de communauté correspond au modèle au sein duquel chaque noeud du réseau est affecté à une seule communauté. Mathématiquement, ce concept s'opérationnalise en représentant la structure en tant qu'une partition d'un graphe, soit pour un graphe $G = (V, E)$ avec k communautés, $G = \{C_1, C_2, ..., C_k\}$.

Structure de communauté hiérarchique La structure de communauté hiérarchique a été introduite en 2002 par Ravasz et Barabási [RB02a]. En effet, il existe certaines situations au sein desquelles il est naturel de représenter les communautés de manière à ce que ces dernières puissent être enveloppées par une communauté parente. Ces structures sont souvent retrouvées dans le contexte de l'analyse de réseaux sociaux. Par exemple, on peut modéliser sous forme de communauté hiérarchique un groupe scolaire, faisant partie d'un niveau, faisant partie d'une école. L'on peut intuitivement s'attendre à ce que les liens entre les invididus membres d'un même groupe soient plus prononcés à l'interne qu'avec les membres des autres classes, mais aussi que les liens entre les membres d'un même niveau soient plus prononcés que les liens avec les membres d'autres niveaux, et ainsi de suite.

Structure de communauté chevauchante Dans [PDF05], un modèle de communauté chevauchante est présenté. En effet, il peut être naturel de représenter une situation au sein de laquelle un élément du réseau à représenter peut faire partie de plusieurs communautés. Il est par exemple fréquent pour un individu de faire partie de plusieurs clubs sociaux, ou encore de faire partie de communautés

qui représentent des situations différentes telles que des situations familiales, amicales, scolaires ou sportives. Il peut être intéressant, dans ces instances, de modéliser la force du lien d'un élément avec une communauté, au lieu de simplement modéliser son appartenance ou sa non-appartenance à une communauté à l'étude.

2.3.2 Méthodes, modèles et algorithmes

Il est possible d'observer des réseaux à l'aide d'observation directe ou de collecte d'information à l'aide de techniques manuelles. Cette méthodologie permet toutefois seulement l'observation de réseaux de petite taille, et l'orientation de ces recherches était généralement basée sur l'étude des propriétés des éléments, ou de petits groupes d'éléments.

L'évolution des méthodes de cueillette de données et la disponibilité de données concernant des réseaux de plus grande taille aura toutefois, dans plusieurs cas, rendu cette méthodologie impraticable. Les scientifiques oeuvrant dans le domaine se sont donc tournées vers des méthodes basées sur l'étude des propriétés statistiques des réseaux [For10].

Sous le régime de la définition «moderne », l'identification des communautés s'opérationnalise en tant que l'analyse du réseau en fonction d'un modèle sous-jacent. Puisque le problème de détection de communautés peut être défini de plusieurs manières en fonction des objectifs de l'utilisateur, plusieurs modèles, techniques et algorithmes ont été développés.

2.3.2.1 Modèles de blocs stochastiques

Le modèle le plus connu est certainement le modèle de blocs stochastique (Stochastic block model - SBM), introduit par Holland, Laskey et Leinhardt [HLL83]. Ce dernier se décrit succinctement ; un modèle est composé de n sommets et de B blocs. La probabilité que les sommets membres d'un même bloc B soient connectés est de p, et la probabilité que les sommets membres de blocs différents soient connectés est q. Si $p > q$, un retrouve une structure associative au sein du réseau. Si $p < q$, on retrouve une structure disassociative. Si $p = q$, on retrouve un modèle aléatoire.

Ce modèle simple et ses extensions a motivé une grande quantité de méthodes de détection de communauté. Une grande quantité de ces méthodes est basée sur la maximisation de la vraisemblance d'observer un réseau sous un modèle donné. Cette mesure se calcule comme

$$\mathcal{L}(G|g) = \sum_{rs} m_{rs} \log \frac{m_{rs}}{n_r n_s} \tag{2.3.1}$$

avec m_{rs} le nombre de liens entre les sommets des groupes r et s, $n_r$et n_s le nombre de sommets dans ces groupes. Il est à noter que G représente le graphe à l'étude et g le paramètre représentant les affectations de noeuds à des blocs.

Une intéressante extension de ce modèle a été introduite par Karrer et al. [KN11] dans l'optique

de prendre en considération la séquence des degrés des noeuds d'un réseau. Cette extension s'écrit

$$\mathcal{L}(G|g) = \sum_{rs} m_{rs} \log \frac{m_{rs}}{k_r k_s} \tag{2.3.2}$$

avec k_r et k_s représentant le degré des noeuds compris dans r et s, respectivement. Cette extension modèle plus fidèlement les réseaux réels au sein desquels la séquence des degrés peut être d'une grande variabilité, et permet d'obtenir des solutions de meilleure qualité pour les réseaux complexes.

Certaines méthodes ont été développées afin de rendre le processus opérationnel même pour des réseaux de grande taille [Pei14]. Toutefois, une lacune de cette méthodologie lorsque comparée à, par exemple, l'optimisation de la modularité (voir 2.3.2.2) est que le nombre de communautés recherchées doit généralement être fourni à l'algorithme à titre de paramètre. Des méthodes de sélection de modèles courantes (AIC ou BIC, par exemple), peuvent être employées pour pallier à ce problème, toutefois, cela complexifie la méthodologie par rapport à des méthodes sans paramètres comme la modularité.

2.3.2.2 Optimisation basée sur un modèle nul

Une méthodologie similaire consiste à maximiser une mesure qui indique à quel point un réseau donné diffère d'un modèle nul, soit un modèle au sein duquel aucune communauté ne devrait exister en théorie.

Modularité La mesure semblable la plus connue et la plus populaire en pratique est certainement la modularité [NG04], définie comme

$$\mathcal{Q} = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(C_i, C_j) \tag{2.3.3}$$

avec m le poids total des arcs du réseau, A la matrice d'adjacence du réseau, k_i et k_j le poids des sommets i et j, et δ la fonction delta de Kronecker, définie comme

$$\delta(i,j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}.$$

La valeur de la modularité est comprise dans l'intervalle $[-0.5, 1]$. Elle est positive si le nombre de liens intracommunautés au sein d'un réseau est plus grand que le nombre de liens que l'on s'attend à retrouver au sein d'un réseau aléatoire. Un réseau avec une modularité de 1 est un graphe complet. En pratique, la modularité offre des résultats très intéressants, ce qui explique sa large adoption par la communauté scientifique. Elle a été utilisée pour trouver des communautés au sein de domaines variés tels que la sociologie et la chimie.

La modularité en tant que mesure comporte toutefois certains problèmes théoriques qui limitent son acceptation par la communauté scientifique. En particulier, une limite de résolution a été identifiée par Fortunato et Bathelemy [FB07]. Cette problématique rend impossible l'identification de communautés à une échelle plus petite qu'un ratio de l'ordre de $\sqrt{m}$. De plus, il a été démontré que la modularité a tendance à trouver des communautés dans certains graphes aléatoires, au sein desquels aucune communauté ne devrait réellement exister [GSPA04].

Plusieurs algorithmes ont été développés pour maximiser la modularité. Le premier d'entre eux est l'algorithme de Newman [New04], qui consiste en un algorithme agglomératif dans lequel les arcs du réseau sont ajoutés itérativement de manière à obtenir la plus grande modularité à chaque étape. Newman introduit ensuite une méthode spectrale pour maximiser la modularité [New06], en proposant une forme de la mesure adaptée à son étude à l'aide d'outils issus de l'algèbre linéaire. La popularité de la mesure s'est grandement accrue avec l'introduction de l'algorithme de Louvain [BGLL08], qui aura permis pour la première fois d'obtenir des résultats de grande qualité à des problèmes de détection de communautés en des temps raisonnables. Cet algorithme est de plus particulièrement intéressant puisqu'il permet de découvrir une hiérarchie de communautés, ce qui, intuitivement, apparait dans plusieurs réseaux naturels tels que les réseaux sociaux.

Surprise La surprise est une mesure similaire introduite avec l'objectif de pallier à certains des problèmes identifiés par la recherche concernant la modularité. Introduite récemment par Aldecoa et Marín [AM11], la surprise représente l'écart entre un réseau observé et un modèle nul basé sur une distribution hypergéométrique. La mesure s'écrit

$$S = -\log \sum_{j=p}^{\min M,n} \frac{\begin{pmatrix} M \\ j \end{pmatrix} \begin{pmatrix} F-M \\ n-j \end{pmatrix}}{\begin{pmatrix} F \\ n \end{pmatrix}} \tag{2.3.4}$$

avec F le nombre maximal de liens dans un réseau avec le même nombre de noeuds, n le nombre de liens observés dans le réseau, M le nombre maximal de liens internes, étant donné la partition considérée, et p le nombre de liens internes observés dans la partition considérée.

Les auteurs expliquent que leur mesure décrit à quel point il est invraisemblable (ou surprenant, ce qui motive le nom de la mesure) d'observer la distribution des arcs et des noeuds du réseau. Il est ensuite démontré expérimentalement à l'aide de réseaux synthétiques que la mesure donne des résultats qui se rapprochent beaucoup plus des résultats attendus en fonction de l'information mutuelle que la modularité, et ce pour un grand nombre d'algorithmes de pointe utilisés fréquemment dans la littérature et en pratique.

Peu d'algorithmes existent pour maximiser cette mesure. Toutefois, il est possible de modifier certains algorithmes à l'origine développés pour maximiser la modularité afin de l'utiliser. On peut

citer en exemple Combo [SCBR14], un algorithme générique développé dans le but d'être compatible avec un grand nombre de mesures de qualité.

WCC La mesure WCC, pour *weighted community clustering*, a été introduite par Prat-Pérez et al. en 2012 [PPDSBLP12]. Cette dernière vise à s'aligner avec certaines caractéristiques que l'on retrouve souvent dans les réseaux sociaux. En effet, les communautés dans les réseaux sociaux ont intuitivement une forte propension à montrer une structure interne avec une forte quantité de triangles. WCC tente de maximiser cette caractéristique au sein des communautés, et cherche aussi à minimiser le nombre de ponts présents dans ces dernières.

La mesure, pour un graphe $G = (V, E)$, s'écrit

$$WCC(S) = \frac{1}{|S|} \sum_{x \in S} WCC(x, S) \tag{2.3.5}$$

avec

$$WCC(x, S) = \begin{cases} \frac{t(x,S)}{t(x,V)} \frac{vt(x,V)}{|S \setminus \{x\} + vt(x, V \setminus S)}. & t(x, V) \neq 0 \\ 0 & t(x, V) = 0 \end{cases} \tag{2.3.6}$$

Dans ces formules, x représente un sommet faisant partie du réseau, S représente le réseau en tant que tel, $t(x, S)$ le nombre de triangles fermés par x avec des sommets contenus dans S, $vt(x, S)$ le nombre de sommets dans S qui composent un triangle avec x.

À ce jour, un seul algorithme existe spécifiquement pour maximiser la WCC, soit l'algorithme introduit par les auteurs de la mesure quelques années suite à son introduction [PPDSLP14]. Cet algorithme est intéressant puisqu'il ne repose pas sur des hypothèses qui rendent son exécution en parallèle difficile. Ce dernier a, par ailleurs, été adapté pour l'exécution sur plates-formes hétérogènes [HVPPLP15].

2.3.2.3 Modèles basés sur des flux sur les arcs

Certains modèles sont basés sur la modélisation de flux sur les arcs du réseau, représentant des échanges ou des relations entre les différentes composantes. Sous ce régime, une communauté consiste en une partie du réseau dans laquelle il y a une forte activité en termes d'échanges ou de liens. En général, les méthodes de détection de communautés basées sur ce modèle utilisent un processus de diffusion sur les arcs du réseau afin de découvrir des groupes de noeuds fortement connectés, l'intuition étant qu'il y aura beaucoup d'activité au sein d'une communauté avec comparativement peu d'activité entre ces dernières. Deux méthodes basées sur ce principe sont revues dans cette section, soit *Walktrap* et *Infomap*.

Walktrap Walktrap a été proposé par Pons et Latapy [PL05]. Cet algorithme est basé sur l'utilisation de marches aléatoires sur un réseau, selon l'intuition dictant qu'un marcheur aura tendance à rester longtemps dans une zone dense d'un réseau, et à en sortir relativement peu fréquemment. Ces zones denses constituent des communautés.

Walktrap définit une mesure de distance entre deux noeuds d'un réseau

$$r_{ij} = \sqrt{\sum_{k=1}^{n} \frac{(P_{ik}^t - P_{jk}^t)^2}{d(k)}} \tag{2.3.7}$$

et similairement la distance entre deux communautés d'un réseau

$$r_{C_1C_2} = \sqrt{\sum_{k=1}^{n} \frac{(P_{C_ik}^t - P_{C_2k}^t)^2}{d(k)}} \tag{2.3.8}$$

Walktrap utilise ensuite une méthode similaire à la méthode de classification hiérarchique de Ward [War63] avec la mesure de distance r afin de détecter des communautés.

Infomap Infomap [RB08] est un algorithme de détection de communautés aussi basé sur l'utilisation de marches aléatoires. Cet algorithme est toutefois basé sur la minimisation d'une mesure nommée « map equation », basée sur la théorie de l'information définie comme

$$L(M) = q_{\curvearrowright} H(\mathcal{Q}) + \sum_{i=1}^{m} p_{\circlearrowright}^i H(\mathcal{P}^i) \tag{2.3.9}$$

avec $q_{\curvearrowright}$ le taux de transition entrantes au sein d'un module q, $p_{\circlearrowright}^i$ la fréquence d'utilisation d'un module p^i et $H(\cdot)$ l'entropie de Shannon [Sha48] d'une distribution.

Cette mesure représente la longueur minimale en bits de la description d'une marche aléatoire de longueur infinie sur le réseau. L'intuition derrière cet algorithme de détection de communautés est qu'il est possible de trouver une description plus courte dans un réseau comportant une structure de communauté. En effet, il est possible d'obtenir une description plus courte en compressant le réseau de manière à regrouper ensemble les noeuds qui communiquent fortement ensemble. La meilleure communauté constitue ainsi la communauté qui minimise $L(M)$.

2.3.2.4 Autres méthodes heuristiques

Certaines autres méthodes de détection de communautés ne peuvent pas être classifiées dans les catégories revues précédemment. Il s'agit, par exemple, de l'algorithme de Girvan-Newman [GN02], historiquement significatif, ou des algorithmes basés sur la propagation d'étiquettes, une méthodologie de détection de communautés intuitive et très rapide.

Algorithme de Girvan-Newman L'algorithme de Girvan-Newman [GN02] est une méthode de détection de communautés divisive basée sur l'utilisation de mesures de centralité. Cet algorithme est particulièrement intuitif. La première étape consiste à, pour tous les arcs, calculer une généralisation de la mesure de centralité nommée *betweenness* [Ant71, Fre77], définie comme le nombre de plus courts chemins passant par un arc. L'arc avec la plus grande *betweenness* est ensuite retiré du réseau. Ces deux opérations sont itérées en succession jusqu'à ce que le graphe devienne disjoint, et la procédure entière est répétée sur chaque composante connexe.

Propagation d'étiquettes La détection de communautés par propagation d'étiquettes [RAK07] est une méthode de détection de communautés très rapide et très simple qui opère directement à l'échelle de la structure du réseau, et n'est pas basé sur l'optimisation d'une valeur. Dans un premier temps, chaque noeud du réseau est initialisé avec une étiquette unique. Ensuite, les noeuds sont visités itérativement dans un ordre aléatoire, et la valeur de leur étiquette prend la valeur la plus fréquemment observée chez les noeuds voisins. Cette procédure est répétée jusqu'à ce qu'il n'y ait plus de changements dans les étiquettes du réseau.

Chapitre 3

Le parallélisme en informatique

Le calcul en parallèle est, dans sa plus simple expression, l'exécution d'un travail informatique sur plusieurs unités d'exécution concurremment. La généralité de cette expression laisse sous-entendre qu'il s'agit d'un concept très large, et c'est tout à fait exact. L'objectif de ce chapitre est de présenter un aperçu général des éléments de base importants qui différencient le calcul en parallèle du calcul séquentiel afin de bien situer le lecteur de ce mémoire.

Dans un premier temps, la section 3.1 présente différentes taxonomies dont l'objectif est de classifier les différents modèles d'exécution en parallèle. La taxonomie de Flynn (sous-section 3.1.1) est particulièrement importante. Par la suite, la section 3.2 présente différentes familles de plateformes matérielles utilisées en calcul parallèle. La section 3.3 présente les deux modes de traitement majeurs utilisés pour structurer les calculs et les algorithmes, et la section 3.4 présente les modes de communication entre les différentes unités de calcul, qui conditionnent la structure des programmes compte tenu de l'importance de la communication pour la performance. La section 3.5 présente certains facteurs et différentes mesures pour évaluer l'efficacité et la performance d'un algorithme parallèle. Finalement, la section 3.6 présente certains problèmes qui affectent les algorithmes conçus pour opérer en parallèle.

3.1 Les taxinomies d'architectures d'exécution parallèle

3.1.1 La taxinomie de Flynn

La taxinomie de Flynn, proposée par Michael J. Flynn en 1966 [Fly66], est une typologie générale qui classifie les différentes architectures d'ordinateurs selon deux dimensions, soit le flux de données et le flux d'instructions. Dans le cadre de cette taxinomie, un flux est défini macroscopiquement comme une séquence d'éléments tel que vu par la machine lors de l'exécution et sur laquelle les unités d'exécution opèrent. Le lecteur peut se référer à [Fly72] pour la description publiée originale de cette classification.

	Instructions	
Données	Simple	Multiple
Simple	SISD	MISD
Multiple	SIMD	MIMD

TAB. 3.1.1: La taxonomie de Flynn

Chacune de ces deux dimensions peut prendre deux valeurs, soit simple ou multiple. Cette taxinomie induit donc quatre architectures, résumées dans le tableau 3.1.1.

3.1.1.1 Single Instruction - Single Data (SISD)

L'architecture SISD réfère à un ordinateur exécutant une instruction à la fois sur un seul flux de données. Il s'agit donc d'un ordinateur exécutant des instructions de manière exclusivement séquentielle. Puisque cette architecture réfère spécifiquement à un mode d'opération qui exclut d'office le parallélisme, elle est de peu d'intérêt dans le cadre de ce travail. Toutefois, la présentation de cette dernière est essentielle à la compréhension de la taxinomie de Flynn, et son étude permet d'asseoir une base de comparaison pour les autres modèles de la taxinomie. À titre d'illustration, un ordinateur avec un seul processeur qui possède un seul cœur a une unité d'exécution unique, et répond donc aux critères qui permettent de le classifier dans la catégorie SISD.

3.1.1.2 Single Instruction - Multiple Data (SIMD)

L'architecture SIMD réfère à un ordinateur possédant plusieurs unités d'exécution, qui exécutent la même instruction appliquée à des données différentes au même moment. Lorsque l'exécution est terminée, les données sont généralement accessibles à toutes les unités d'exécution afin qu'elles puissent être utilisées à titre de résultat intermédiaire par des instructions subséquentes [Dun90].

Les unités d'exécution dans une machine dont l'architecture s'apparente à la classe SIMD sont souvent logiquement représentées comme étant arrangées dans un vecteur ou un tableau dont les éléments sont interconnectés [Sve92]. Historiquement, la première implémentation matérielle basée sur ce principe découle du projet ayant mené à la construction de l'ILLIAC IV [BBK+68], considéré comme le premier superordinateur. L'attention des chercheurs s'est ensuite tournée vers le développement de processeurs, avec par exemple le DAP [Par83] et le MPP [Bat80, Pot85], deux processeurs antiques ayant été conçus pour opérer sous cette architecture. Finalement, le projet LUCAS [FKS86], digne de mention, était l'un des premiers projets universitaires dont l'objectif était le design et l'évaluation d'un système massivement parallèle. Un nombre d'exemples historiques supplémentaires est détaillé dans le travail de Svensson précité [Sve92].

Bien qu'une architecture SIMD impose des contraintes qui rendent la tâche de développer des programmes pour ce genre d'architecture plus fastidieuse, cette dernière est particulièrement adaptée à certains types de problèmes décrits comme « data parallel » (voir la section 3.3). Par exemple, la

computation d'images et de graphiques constitue un problème qui entre dans cette catégorie, et l'importance pratique et commerciale de ce problème a motivé le développement de processeurs spécialisés de plus en plus puissants. La réalisation que le modèle SIMD tel qu'implémenté à faible cout par ces processeurs spécialisés a aussi des applications en calcul scientifique a par la suite à son tour motivé le développement de plateformes de calcul comme OpenCL et CUDA (voir la sous-section 3.2.2), qui sont particulièrement appréciées par les chercheurs dans différents domaines scientifiques.

3.1.1.3 Multiple Instruction - Single Data (MISD)

L'architecture MISD réfère à une architecture qui exécute des instructions différentes sur le même flux de données. Certains auteurs doutent que ce type d'architecture parallèle n'ait jamais existé [Bar16a], tandis que d'autres affirment que certains exemples pourraient être, par exemple, des systèmes redondants tel que le système de contrôle de vol de la navette spatiale américaine [GPHB09], ou encore certains systèmes de reconnaissance de formes configurés pour reconnaitre de multiples formes en même temps à partir du même ensemble de données [HSN+04].

3.1.1.4 Multiple Instruction - Multiple Data (MIMD)

L'architecture MIMD [Buz85] réfère à un système possédant plusieurs unités d'exécution autonomes, qui sont en mesure d'exécuter des instructions différentes sur des ensembles ou des flux de données distincts.

Cette famille typologique constitue la plus fréquemment rencontrée en pratique. Par exemple, un ordinateur avec plusieurs microprocesseurs, ou bien un processeur multicœur, est un exemple d'architecture MIMD. Une grappe de serveurs (cluster), qui constitue un réseau d'ordinateurs interconnectés généralement utilisés conjointement afin de résoudre un problème de grande taille, est un autre exemple d'architecture qui répond à la définition de la classification MIMD.

L'architecture MIMD peut généralement être séparée en deux sous-classes, en fonction de la manière avec laquelle les processeurs accèdent à la mémoire [Dun90]. Cette dernière peut être centralisée ou décentralisée.

Dans le contexte centralisé, la mémoire est partagée entre tous les processeurs, qui y accèdent de la même manière. Dans un ordinateur avec un processeur multicœur, les unités d'exécution accèdent généralement à la mémoire de cette manière. Ce mode d'accès est expliqué dans la sous-section 3.4.1.

Dans le contexte décentralisé, la mémoire peut être partagée entre les processeurs, auquel cas l'accès à cette dernière s'effectue de manière non uniforme, ou non, auquel cas l'application doit se charger de transférer les données par un autre canal. Dans tous les cas, dans le contexte décentralisé, la localisation des données par rapport aux unités d'exécution est un enjeu de performance majeur. Les grappes de serveurs accèdent généralement à la mémoire de cette manière. Ce mode de communication est expliqué dans la sous-section 3.4.2.

3.1.2 La taxinomie de Feng

La taxinomie de Feng [Fen72] est basée directement sur la dichotomie entre la computation en série et le calcul en parallèle. Cette taxinomie propose de classifier les architectures informatiques en fonction du degré de parallélisme que ces dernières sont en mesure d'exhiber. Le degré maximal de parallélisme est défini comme étant le nombre maximal de bits avec lesquels un ordinateur peut effectuer des calculs en un temps donné.

Comme Flynn, Feng classifie les systèmes informatiques sur deux dimensions, soit selon les mots (qui constituent la plus petite unité de données sur laquelle un processeur peut travailler) et les bits. Ces deux dimensions sont caractérisées par leur degré de parallélisme maximal respectif. Bien que cette classification soit intéressante d'un point de vue historique et théorique, elle ne donne que peu d'information utile en pratique dans un contexte moderne.

La classification permet de représenter le parallélisme d'une plateforme sous la forme d'une paire (b,w), où b est le nombre de bits sur lequel des calculs peuvent être effectués en même temps au sein d'un mot (degré maximal des bits), et w est le degré maximal de parallélisme des mots. Cette taxinomie induit une classification générale, mais cette dernière est maintenant quelque peu désuète :

Word Serial - Bit Serial (WSBS) Le calcul s'effectue exclusivement sur un bit à la fois. Il s'agit d'une forme extrême de calcul en série, que l'on ne retrouve pas en pratique.

Word Parallel - Bit Serial (WPBS) L'ordinateur opère sur plusieurs mots à la fois, mais ces calculs opèrent sur un seul bit à la fois.

Word Serial - Bit Parallel (WSBP) L'ordinateur opère sur un seul mot à la fois, mais les instructions peuvent s'opérer sur plusieurs bits en même temps au sein de ce mot.

Word Parallel - Bit Parallel (WPBP) Il s'agit d'un ordinateur pleinement parallèle, en mesure d'effectuer des calculs sur plusieurs mots en même temps, et d'agir sur plusieurs bits au sein de chacun de ces mots.

3.1.3 La taxinomie de Handler

Cette taxinomie, proposée par Wolfgang Händler [Hän86], étend et est beaucoup plus utile que celle de Feng présentée dans la section précédente. Cette dernière vise à identifier le degré de parallélisme inhérent au matériel informatique qui compose un ordinateur selon trois dimensions, soit :

- Processor Control Unit (PCU)
- Arithmetic Logic Unit (ALU)
- Elementary Logic Circuit (ELC)

La dimension PCU correspond essentiellement à un processeur physique. La dimension ALU correspond à une unité de calcul. La dimension ELC correspond aux circuits physiques nécessaires afin d'effectuer des calculs sur un bit. Chaque PCU peut contenir plusieurs ALU effectuant la même opération concurremment. Les ALU contiennent de multiples ELC, qui sont eux dédiés au calcul d'une position de bit. De cette manière, le parallélisme d'un ordinateur peut être donné sous la forme d'un triplet (k, d, w), où k est le nombre de PCU, d le nombre d'ALU et w le nombre d'ELC.

3.2 Les plateformes matérielles

3.2.1 Central Processing Unit (CPU)

Le CPU réfère au microprocesseur « standard » qui accompagne un ordinateur. Ce dernier inclut en général de la mémoire, des registres, une unité logique et arithmétique et une unité de contrôle. Cet ensemble de composantes constitue l'architecture Von Neunmann [VG93].

Le CPU exécute des instructions qui font partie d'un ensemble d'instructions. On réfère généralement à cet ensemble d'instructions comme étant l'architecture du microprocesseur (Instruction Set Architecture, ou ISA). La plus commune de nos jours est l'architecture x86 (x64), utilisée par les plus grands manufacturiers de microprocesseurs que sont Intel et AMD.

Les architectures sont généralement classifiées comme faisant partie de la famille ayant un ensemble d'instructions complexes (CISC) et les architectures ayant un ensemble d'instructions réduit (RISC). La différence principale entre les deux familles est que les architectures CISC possèdent un ensemble d'instructions beaucoup plus riche, qui permettent de créer des programmes en assembleur beaucoup plus simplement. Il est à noter que la famille RISC est beaucoup plus populaire, et inclut l'architecture dominante x86.

Le développement de microprocesseurs toujours plus performants a suivi une tendance appelée la loi de Moore [Moo06]. Cette dernière dicte que le nombre de transistors que comprend un microprocesseur double à un intervalle approximatif de deux ans. Jusqu'au début des années 2000, cette loi était aussi applicable à la vitesse de l'horloge des processeurs, qui régule la fréquence à laquelle des instructions peuvent être exécutées en série. Ce n'est toutefois plus le cas de nous jours, puisque l'horloge des processeurs moderne opère à une vitesse tellement grande qu'elle est difficile à accélérer en miniaturisant les composantes électroniques.

Les concepteurs de processeurs se sont donc tournés vers le calcul concurrent afin de continuer à produire des microprocesseurs plus performants. Ces derniers ont donc développé des microprocesseurs multicœur, qui consistent en fait en plusieurs processeurs sur la même puce, partageant certaines composantes, dont leur mémoire cache dans le but de rendre la communication entre les différentes unités de calcul plus rapide.

Finalement, on doit mentionner que les concepteurs de microprocesseurs ont développé des extensions à l'ensemble d'instructions x86 basées sur le paradigme SIMD afin de permettre aux déve-

loppeurs d'applications d'accélérer le calcul dans leurs programmes lorsque ce dernier se prête bien à l'exploitation du parallélisme. La première de ces extensions a été MMX (Matrix Math Extensions) par Intel, un ensemble d'instructions qui a ensuite été étendu par AMD avec 3DNow !. Intel a ensuite développé SSE (Streaming SIMD Extensions) at AVX (Advanced Vector Extensions).

3.2.2 Graphics Processing Unit (GPU)

Le GPU est un coprocesseur qui est, de manière typique, utilisé pour effectuer générer des images qui sont par la suite présentées à l'utilisateur sur un écran. Ces derniers sont particulièrement adaptés au calcul impliquant des vecteurs et des matrices. Le développement de programmes effectuant des calculs scientifiques sur GPU était toutefois particulièrement difficile à l'origine puisque le problème à résoudre devait être transformé de manière à se retrouver sous forme graphique.

Pour résoudre ce problème, NVIDIA, un des principaux manufacturiers de GPU, a développé une API spécialisée nommée CUDA, qui permet de développer des logiciels en C sans avoir à transformer le problème visé par le programme sous forme graphique. Cette API est la plus utilisée en pratique, mais elle est propriétaire et ne fonctionne par conséquent qu'avec les GPU produits par NVIDIA. Une API plus générale, OpenCL [SGS10], est développée par le groupe Khonos. Cette API permet de développer des programmes qui sont en mesure d'être exécutés sur une multitude de plateformes d'exécution parallèle, incluant des microprocesseurs multicœur standard, et des GPU manufacturés par NVIDIA, AMD, Intel et autres. Il semble toutefois que l'API CUDA, plus mature que OpenCL, soit plus performante malgré son manque de généralité [FVS11, DWL+12]. Un standard similaire intéressant consiste en OpenACC, qui permet de transformer le code en code accéléré qui s'exécute sur un ou des GPU ou possiblement sur d'autres accélérateurs à l'aide d'annotations [WSTA12, XCC13]. OpenACC utilise de cette manière le même modèle que OpenMP [DM98], un standard similaire pour CPU.

Les GPU sont particulièrement utiles pour exécuter des programmes développés selon un paradigme nommé stream processing, qui consiste essentiellement à définir des fonctions à appliquer concurremment sur des séquences de données indépendantes. La section 3.3.2 de ce travail traite du stream processing plus en détail.

3.2.3 Field Programmable Gate Array (FPGA)

Un FPGA [KTR07] est un circuit programmable qui peut être configuré par l'utilisateur. Lors de la configuration, le FPGA est physiquement reconfiguré afin d'exécuter une application déterminée. De cette manière, il est possible de les configurer afin d'exécuter un nombre impressionnant d'opérations en même temps. En calcul scientifique, ces appareils sont essentiellement utilisés à titre de coprocesseurs, un peu comme un GPU, mais sans exhiber la même flexibilité.

Même s'ils peuvent accélérer largement le traitement des données, ces appareils sont toutefois plutôt rarement utilisés en pratique. Une utilisation recensée concerne leur utilisation en forage de

bitcoins [BC15]. Microsoft les utilise aussi afin d'accélérer des algorithmes de data mining utilisés dans le cadre de son moteur de recherche [PCC+15].

3.2.4 Application-Specific Integrated Circuit (ASIC)

Un ASIC est un circuit construit spécifiquement pour effectuer une tâche dédiée. Leur utilisation s'apparente à celle qui est faite des FPGA, toutefois, les ASICs ne sont pas reprogrammables. Leur utilisation est très marginale en recherche pour le moment. Une utilisation recensée est, encore une fois, leur utilisation pour le forage de bitcoins [Tay13], ou bien leur utilisation pour implémenter un réseau de neurones dans un contexte de vision par ordinateur [FMA+10].

3.3 Modèles de traitement et de calcul

3.3.1 Le traitement par lots (Batch Processing)

Le traitement par lots, ou batch processing, est un modèle de programmation au sein duquel le programmeur sépare un jeu de données en plus petits morceaux afin de les distribuer à des unités de travail indépendantes.

Le traitement par lots est un modèle employé fréquemment pour le traitement de jeux de données de très grande taille (big data). Sa popularité s'est fortement accrue au courant des dernières années, avec le lancement public de la technologie MapReduce [DG04] par Google. Dans ce modèle, les données sont segmentées et distribuées à des unités de travail (étape map), qui effectuent des calculs et obtiennent des résultats intermédiaires. Ces résultats intermédiaires sont ensuite combinés afin d'obtenir un résultat final (étape reduce). Plusieurs implémentations de ce modèle existent, la plus populaire étant Hadoop [Whi15], un projet Apache lancé officiellement en 2011.

3.3.2 Le traitement par flux (Stream Processing)

Le traitement par flux, ou stream processing, est un modèle de programmation qui permet à un programmeur de modeler son application de manière à bien découpler les dépendances entre les tâches et les données, ce qui permet de rendre le calcul parallèle beaucoup plus aisée. Le traitement par flux est une condition nécessaire pour la programmation visant l'exécution sur des GPU modernes [Mac03].

Le traitement par flux consiste à créer des fonctions, nommées kernels, qui sont appliquées sur des flux de données. Cette méthodologie propose deux avantages majeurs. Premièrement, elle permet de traiter des données sans les stocker, un avantage majeur pour les jeux de données de très grande taille. Deuxièmement, elle explicite les dépendances entre les données et les tâches, ce qui rend l'identification de tâches à haut cout, tel que le chargement des données d'un espace mémoire à un autre, relativement aisé. De plus, structurer un programme de cette manière permet d'exploiter au maximum l'architecture SIMD, en exploitant le *pipelining* d'un GPU, par exemple.

On peut représenter mentalement un programme développé avec cette architecture sous la forme d'un graphe, au sein duquel les *kernels* sont représentés par les nœuds et les flux de données par les arcs. Cette représentation permet de visualiser facilement les dépendances entre les tâches et les données. Les données traversent ce graphe d'exécution indépendamment lors du traitement.

On peut faire un parallèle entre le traitement par flux et le modèle de programmation fonctionnel, proposé par Backus [Bac78]. Dans son argumentaire, Bakus proposait qu'il serait plus facile de structurer des programmes sous la forme d'une composition d'une multitude de fonctions sans effets de bord, ce qui permet d'aisément découpler les dépendances entre les données entre les différentes fonctions qui composent un programme. Ce modèle de programmation obtient beaucoup d'attention de la part des développeurs de logiciels depuis quelques années, spécialement en calcul parallèle ; s'il n'y a pas de dépendance entre deux fonctions, ces dernières peuvent être exécutées simultanément de manière sécuritaire par défaut.

3.4 Modèles de communication en calcul parallèle

Dans la grande majorité des algorithmes parallèles, les différentes unités de calcul doivent être en mesure de communiquer entre elles à certaines étapes de l'algorithme afin de synchroniser le travail effectué. Deux modèles majeurs de communication existent, soit le modèle de communication par mémoire partagée et le modèle de communication par échange de messages.

3.4.1 Communication par mémoire partagée (Shared Memory)

Dans ce modèle, plusieurs tâches ont accès au même espace mémoire de manière concurrente. Les unités d'exécution sont donc en mesure d'avoir accès directement aux mêmes données, et aussi par conséquent de communiquer de l'information entre elles respectivement en écrivant et en lisant à la même position en mémoire ; ce mode d'accès est donc très rapide.

À haut niveau, ce modèle simplifie la programmation des algorithmes. Toutefois, il introduit aussi certains problèmes. Par exemple, si deux processus écrivent au même endroit dans la mémoire sans en obtenir l'accès exclusif pour la durée de l'opération, les données peuvent être laissées dans un état inconsistant[1]. Un certain niveau de synchronisation est donc requis afin de coordonner les différents processus et les différents fils d'exécution (threads) qui partagent le même espace.

3.4.2 Communication par échange de messages (Message Passing)

Dans ce modèle, les différentes tâches ont accès à un espace de mémoire locale qui leur est réservé. Les tâches ne peuvent pas accéder à la mémoire disponible aux autres tâches. Ces dernières doivent

1. Pour certaines architectures, cette affirmation n'est pas strictement vraie. Par exemple, l'architecture x86 offre certaines opérations qui permettent d'implémenter des fonctionnalités communes telles que `load`, `store`, `exchange`, `fetch_and_add` de manière atomique.

donc envoyer et recevoir des messages afin d'échanger de l'information lorsque cela est nécessaire pour synchroniser leur calcul.

En calcul scientifique, l'implémentation la plus fréquemment rencontrée est MPI (Message Passing Interface) [Bar16b]. Cette dernière est une spécification basée sur le consensus du MPI Forum, un regroupement industriel chargé de piloter le projet. Cette spécification permet à différentes composantes de communiquer de manière standardisée.

3.5 Facteurs de performance en calcul parallèle

3.5.1 L'intensité de calcul (Computational Intensity)

L'intensité de calcul est définie comme étant le nombre d'opérations de calcul par opération d'accès à la mémoire. Ce concept est important ; dans bien des instances de calcul parallèle, le temps de calcul est dominé par le temps requis pour transférer des données dans la mémoire locale de l'unité d'exécution, et par le temps requis pour synchroniser des résultats intermédiaires entre les différentes unités d'exécution. La problématique ici est due au fait que les couts de communication entre les différentes unités d'exécution augmentent généralement avec le nombre d'unités d'exécution, ce qui réduit l'efficience du parallélisme. Améliorer l'intensité de calcul permet par conséquent de passer plus de temps à effectuer des calculs importants et moins de temps sur des tâches accessoires.

3.5.2 Le parallélisme inhérent aux données (Data Parallelism)

Le parallélisme inhérent aux données est défini comme étant la possibilité d'effectuer des opérations sur une séquence de données de manière indépendante d'autres séquences [HS86]. Le parallélisme inhérent aux données est en dichotomie avec le parallélisme inhérent aux tâches, qui consiste en l'exécution de tâches différentes sur plusieurs unités d'exécution au même moment, mais en utilisant le même ensemble de données. Dans la taxinomie de Flynn, le premier type de parallélisme s'apparente au modèle SIMD, tandis que le deuxième au modèle MISD.

3.5.3 La localité des données (Data Locality)

La localité des données est l'agencement des données de manière à pouvoir avoir accès à ces dernières de manière plus rapide en plaçant les données les plus chaudes, soit celles sur lesquelles le plus de calculs sont effectués, plus près des unités d'exécution [Den05].

La localité des données est spécialement importante dans le cadre de calculs effectués sur GPU ou à l'aide de système distribués. En effet, la performance d'un algorithme exécuté sur une de ces plateformes est souvent directement reliée à la localité des données [KM92].

La localité des données est aussi importante pour améliorer la performance d'algorithmes qui opèrent sur microprocesseurs traditionnels ; transférer des données de la mémoire de l'ordinateur

vers le processeur est beaucoup plus difficile que de transférer des données à partir du cache disponible sur le circuit. Plusieurs techniques ont été développées afin de rendre des programmes plus efficaces vis-à-vis la localité des données. Toutefois, leur utilisation complexifie souvent indument le programme, et il faut faire attention à ne pas optimiser l'algorithme prématurément [Nys14]. Le même principe s'applique pour des algorithmes qui s'exécutent sur systèmes distribués et sur coprocesseurs (GPU ou autre).

3.6 Problématiques touchant le calcul parallèle

Deux problématiques sont rencontrées particulièrement fréquemment en calcul parallèle. La première est nommée *situation de compétition* (race condition) et la deuxième *interblocage* (deadlock).

3.6.1 Situation de compétition (Race Condition)

Une situation de compétition [NM92] est une problématique fréquente au sein d'algorithmes parallèles qui communiquent à l'aide de mémoire partagée. Le problème est rencontré lorsque par deux processus accèdent à la même ressource sans être adéquatement synchronisés. Ces processus peuvent alors entrer en compétition pour la ressource et y accéder dans un ordre qui n'était pas prévu au préalable, ce qui peut corromptre les résultats.

Pour illustrer, prenont un algorithme au sein duquel deux processus A et B tentent d'incrémenter une valeur $x = 0$ concuremment. Si les processus sont bien synchronisés, les opérations s'exécuteraient dans cet ordre :

Étape	Procesus A	Processus B	Valeur de x
1	Lire x		0
2	Incrémenter x		1
3		Lire x	1
4		Incrémenter x	2

Dans une situation de compétition, cet ordre pourrait être :

Étape	Procesus A	Processus B	Valeur de x
1	Lire x		0
2		Lire x	0
3	Incrémenter x		1
4		Incrémenter x	1

Les situations de compétition peuvent être évitées par l'emploi d'opérations atomiques offertes par la plateforme matérielle pour des opérations simples, ou par l'utilisation de primitives de synchronisation telles que des exclusions mutuelles (mutex), qui permettent à un processus d'obtenir l'accès exclusif à une ressource. Ces dernières primitives doivent toutefois être utilisées avec soin puisqu'elles introduisent un autre problème, soit l'interblocage.

3.6.2 Interblocage (Deadlock)

L'interblocage est un problème décrit par Edward G. Coffman Jr. en 1971 [CE71]. Ce problème survient dans le cadre d'un algorithme parallèle lorsqu'une tâche est mise en attente d'une ressource présentement monopolisée par une autre tâche, qui est elle-même en attente d'une ressource qui dépend de la première tâche. L'on se retrouve alors dans une situation de dépendance circulaire, et le travail effectué par l'algorithme est interrompu.

Une solution pour faire face à ce problème consiste en la confection d'algorithmes sans blocage. Ces derniers sont classifiés selon trois catégories, qui offrent des garanties différentes :

Algorithmes sans attente (Wait-Free) Un algorithme sans attente [Her91] est un algorithme qui garantit que n'importe quel processus est en mesure de compléter son exécution, sans considération pour les autres processus qui pourraient opérer en même temps.

Algorithmes sans verrous (Lock-Free) Un algorithme sans verrous [MP92] est un algorithme qui garantit qu'au moins un processus de l'algorithme puisse progresser, et ce, à n'importe quel point de l'exécution.

Algorithmes sans obstructions (Obstruction-Free) Un algorithme sans obstructions [HLM03] est un algorithme qui garantit qu'un processus va progresser dans son exécution s'il est isolé des autres processus qui opèrent concurremment.

Chapitre 4

Parallel Methods for Community Detection in Complex Networks

Philippe Gagnon, Gilles Caporossi, Sylvain Perron

Abstract Community detection in networks is an important problem with applications in various scientific fields, including sociology, business, informatics, engineering, biology and chemistry. Most state-of-the-art methods rely on the assumption that the algorithm will be executed in a sequential manner, which can be problematic as the community detection problem is $\mathcal{NP}$-complete under most of its accepted definitions and approximations. As the size of networks under study grows, faster methods must be developed in order to obtain results in reasonable amounts of time. In this work, we develop and implement parallel community detection methods based on current state-of-the-art fast methods and evaluate their effectiveness against natural and benchmark graphs.

Keywords Complex Networks, Community Detection, Parallel Algorithms

4.1 Introduction

A network (graph) is defined as a set of elements, called vertices or nodes, related to each other by connections named links or edges. The study of these structures is a field of mathematics named graph theory, which is studied since Euler introduced it in 1736 in his celebrated solution to the Königsberg Bridge Problem [Eul41, Shi12].

More formally, a graph $G = (V, E)$ is a pair of sets where V is the set of vertices in the graph, and $E = \{(x, y) \subseteq V^2\}$ is the set of edges denoting relations between the vertices. A graph can be *directed*, in which case E is a set of ordered pairs, or *undirected*, in which case this information is not available. Properties are often attached to vertices and edges. In that aspect, it is worth mentioning

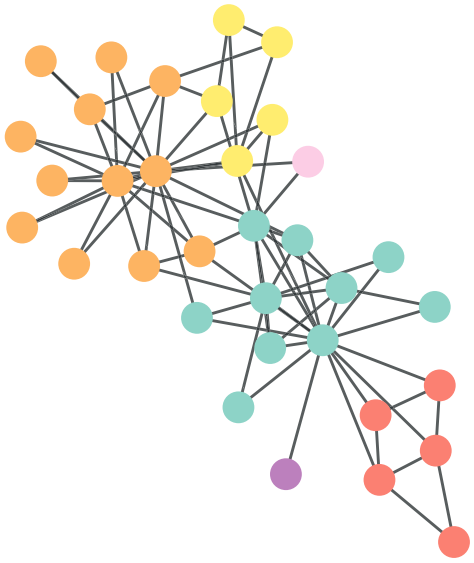

Figure 4.1.1: Visualization of the community structure of Zachary's Karate Club Network [Zac77], as detected by modularity maximization using simulated annealing. In this case, modularity maximization detected four communities and two unassigned vertices (which technically form singleton communities, but in our view this interpretation would be improper from a practical standpoint). This is in contrast with the ground-truth communities provided by Zachary, in which only two communities are present.

that weights are often attached to edges, denoting the importance or magnitude of the relation they represent in the network. Networks in which edges have this property are called *weighted networks*.

Many complex systems which are found in the real world can be modeled as networks and further studied using techniques derived from graph theory [MMG+07, LAH07, FLG00, PDF05]. Since these networks model complex systems, they are called *complex networks* in the literature. Interestingly, these networks generally have properties which cannot be found in simpler graphs, such as regular lattices or random graphs. In this work, we focus on a property, inherent to some complex networks, known as *community structure*. In [RCC+04], Radicchi et al. further refine the community structure definition and introduce definitions of communities in a weak and strong sense.

Community structure (see figure 4.1.1 for an example) is present in networks where vertices can be grouped such they are densely connected to other vertices that are members of the same community, while being sparsely connected with vertices that are members of other communities. Determining the community structure of a complex network is known as *community detection*.

Many methods and algorithms have recently been proposed for community detection. However, under most of its generally accepted definitions, community detection is a $\mathcal{NP}$-hard problem. Efficient algorithms to find exact solutions are thus not currently available, and even heuristics currently have difficulty finding high-quality solutions in reasonable amounts of time. Efficient parallel algo-

rithms would offer an opportunity to tackle the larger problems which are becoming more common today, but most mainstream methods are of inherently sequential nature and exhibit a high degree of data dependency. As such, executing these algorithms in parallel is a non-trivial task. This is the problem on which we focus in this work. Our contributions are as follows:

1. We further explain the challenges related to parallel community detection in complex networks;
2. We present parallel implementations of community detection algorithms inspired from major community detection methods from the relevant literature;
3. We evaluate and benchmark the proposed algorithms using graphs representing artificial and real-world data.

4.2 Methods for Community Detection in Networks

This section reviews a selection of the more popular community detection methods that have been recently proposed by various researchers. Many methods for community detection have been proposed in recent years. While some of them are based on unique or novel principles, the main current methods can be grouped in three different families:

1. Methods based on block models;
2. Methods based on null models;
3. Methods based on flow models;

Since so many community detection methods have been proposed in recent years, the aim of this section is not to provide the reader with a comprehensive overview but to outline the main methods on which the current state of the art is based. For a more in-depth treatment, the reader is referred to [For10], which is an excellent review of the field as of 2010, or to [CGP11] for an alternative perspective. One could also refer to [FH16] for a more recent albeit more limited treatment.

4.2.1 Methods Based on Block Models

Block models were first introduced in [WBB76] for the study of social networks. In that work the authors proposed block models as a tool to represent the ties, roles and positions between different individuals in social contexts. This methodology was further extended in [HLL83], in which a stochastic generalization of the prior work, the stochastic block model, was introduced. In the stochastic block model, vertices are partitioned (grouped in blocks) in such a way that the probability of an edge being present between two vertices is dependent on their respective block memberships. Community assignments can thus be recovered by maximizing the likelihood of observing a graph G with respect to community assignments g.

An extension to this model was proposed by Karrer and Newman [KN11] in order to account for the variation in the degree of the vertices composing the graph. This model appears to give more realistic results in practice and is in usage by state-of-the-art community detection methods such as [Pei14]. Interestingly, [New16] recently demonstrated an equivalence between the degree-corrected stochastic block model and modularity maximization described in subsection 4.2.2.

4.2.2 Methods Based on Null Models

A similar methodology consists of maximizing a measure that indicates to which extent a network differs from a null model, in which no community should in theory exist. The most popular measure based on this principle is named modularity [NG04].

Modularity is a quality function for community detection. It describes how much more densely connected a subgraph appears as compared to how densely it would be connected in a null model (usually a Erdös-Rényi random graph [ER59]). The measure can be written down as follows:

$$\mathcal{Q} = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(C_{i,} C_j) \tag{4.2.1}$$

with m the total link weight in the network,A the adjacency matrix of the graph, k_i, k_j the degree (or weight for weighted networks) of nodes i and j, C_i and C_j the current communities of nodes i and j, and δ the Kronecker delta function, which is defined as

$$\delta(i, j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}. \tag{4.2.2}$$

Shortly after its introduction, researchers have found and highlighted theoretical issues with modularity as a quality measure for community detection. In particular, [FB07] highlights a resolution limit, and [GSPA04] highlights that modularity can often point to community structure even in random graphs (where no such structure should exist), which is quite undesirable. Nevertheless, modularity maximization detects high quality communities in practice, and is one of the most popular community detection methods in practical use today.

Algorithms based on modularity maximization are numerous. Newman [New06] proposed a spectral algorithm based on the eigenvectors of the «modularity matrix». Blondel, Vincent and Guillaume [BGLL08] proposed the Louvain algorithm, a greedy agglomerative procedure which is able to detect hierarchical communities. It is considered one of the best and fastest community detection methods in the literature. Their method has been extended by [DP15] to directed networks.

Other methods include [RB06], which proposes a method based on minimizing a Potts spin model using simulated annealing. In [ACC+10], Aloise et al. proposed an exact modularity maximization algorithm that is able to obtain the exact solution for problem instances with up to 512 entities,

which is impressive considering the hardness of the problem.

4.2.3 Methods based on Flow Models

Another family of models attempts to model flows on a network instead of its topological structure. Intuitively, in a network displaying community structure, that is, groups of nodes densely connected internally, and sparsely connected externally, one would assume that flows between vertices would tend to stay within communities most of the time, and scarcely move between communities. Since community structure is discovered by following edges between vertices, these algorithms are particularly well suited to the study of directed networks.

Of note, the map equation [RAB09] is an information-theoretical measure based on this principle. Conceptually, it describes the theoretical lower bound of the length of the description of a random walk on a network. Under this framework, finding the best community structure is equivalent to finding the group assignments which minimize the equation. The prominent algorithm based on this principle is Infomap [RB08], which has recently been extended to the hierarchical [RB11] and overlapping cases [ER11].

Another algorithm based on random walks on graphs is the one by Pons and Latapy [PL05], named Walktrap. In that work the authors proposed a distance measure r between vertices based on the probability of a random walker walking from a vertice to another in a fixed number of steps. Using r, communities are obtained by clustering vertices with a method akin to Ward's [War63].

4.2.4 Other Heuristics

Girvan-Newman Algorithm The algorithm by Girvan and Newman [GN02] is one of the more interesting from a theoretical and historical point of view, because it introduced the field to physicists who started using statistical mechanics to study complex networks [RB02b]. In this algorithm, betweenness centrality is computed for all edges, and edges are removed according to their centrality score. This process is repeated iteratively. A community is obtained when a component is disconnected from the graph. This method is still in use today, but in practice it is quite slow, running in $\mathcal{O}(mn^2)$, and thus limited to the study of smaller graphs.

Label Propagation Label propagation algorithms have been employed in order to detect communities in networks. The original algorithm (LPA), proposed by Raghavan, Albert and Kumara in [RAK07], works by assigning a label to each node in the network. At each iteration of the algorithm, nodes are considered in random order and their label is updated to the label that is most commonly found among its adjacent nodes. This algorithm is extremely scalable as it runs in near-linear time. However, it is not stable and often produces partitions of lower quality than competing methods. In [BC09], Barber and Clark extended LPA by framing it as a constrained optimization problem. They propose multiple constrained LPA algorithms, including LPAm (a modularity-specialized LPA), LPAb

(a modularity-specialized LPA for bipartite networks) and LPAr, which modifies the tie-breaking rule of the original LPA algorithm, improving its stopping conditions. In [LM10], Liu and Morata proposed LPAm+, which extends LPAm by introducing a merging step, which allows the algorithm to escape some local minima that are reached due to LPAm being biased in favor of communities of similar sizes.

4.3 Parallel Community Detection

In this section, we explain the parallel community detection problem and review prior works in the field.

4.3.1 Problem Statement

Most of the mainstream state-of-the-art community detection methods as described in section 4.2 are iterative sequential heuristics. Their convergence guarantees are dependent on the iterative nature of the optimization process that defines communities. The fastest algorithms available have superlinear complexity (e.g. Louvain). This would not be an issue if computation resources and time were unlimited, but in practice that is never the case. Since the size of available datasets keep increasing faster than sequential speed improvements on modern processors, algorithms that perform well on parallel architectures must be developed in order to address today's challenges.

4.3.2 Concrete Issues

The major issues caused by parallel execution in community detection algorithms are mainly caused by race conditions. A race condition is an issue that arises when two or more workers are writing to a shared memory location concurrently. When instructions are executed in parallel, it is not possible to determine the order of the execution. In our case, as stated earlier, most state-of-the-art community detection algorithms consist of inherently sequential, iterative heuristics. Race conditions thus often break the assumptions on which community detection algorithms are based. Three major issues can be identified. They are discussed in the following paragraphs.

Node Swapping The node swapping issue can occur when two vertices or more are evaluated at the same time by two or more distinct workers [BHW+13]. Consider a vertex a which is a member of community C_1. This vertex is evaluated and must move to community C_2. Further consider a vertex b currently in community C_2, which is concurrently evaluated and must move to community C_1 in order to improve the community structure at that stage. In most graph structures, this kind of node swapping will not improve the community structure that is being sought. A simplified illustration of this phenomenon is located in figure 4.3.1.

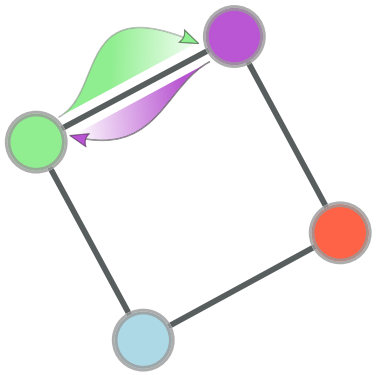

Figure 4.3.1: Node swapping in a simple graph. If the vertex in the green community decides to move to the purple community, while the vertex in the purple community decides to move to the green community during the same iteration, no change has actually happened to the community structure of the network. This is a toy problem with no consequence, but these moves do happen during community detection procedures when working with information that is not quite up to date.

Negative Improvement As stated in the introduction, community detection algorithms are generally sequential, iterative procedures. It is usually guaranteed at every algorithm iteration that the community structure according to some target will improve, by virtue of rejecting moves at each stage that would deteriorate the overall solution. Since moves are evaluated in isolation, this property guarantees that the algorithm will improve the community structure in a monotonic fashion.

In the parallel case, it cannot be guaranteed that changes in the community structure will be evaluated in isolation. Thus, the assumption that community structure will improve monotonically cannot be preserved in the usual fashion.

To illustrate, take the modularity formula 4.2.1, for which an update corresponds to

$$\Delta Q_{i\to C(j)} = \frac{\sum_{j\in C(j)}\omega(i,j) - \sum_{k\in C(i)\setminus\{i\}}\omega(i,k)}{m} + \frac{\omega(i)\times\omega(C_{i\setminus\{i\}}) - \omega(i)\times\omega(C_j)}{2m^2}. \quad (4.3.1)$$

In [LHK15], Lu et al. show that, when moving vertices i and j concurrently to a community $C(k)$, $\Delta Q_{i\to C(k)} + \Delta Q_{j\to C(k)} < \frac{\omega(i)\times\omega(j)}{2m^2} \Rightarrow \Delta Q_{\{i,j\}\to C(k)} < 0$. This case can notably arise where there is no edge between i and j.

Stopping Conditions State-of-the-art community detection methods are generally iterative procedures which terminate when no improvement can be obtained by running the algorithm further. A popular stopping condition, used in particular with algorithms based on label propagation, is to terminate when no moves have occurred during an entire iteration.

In the parallel setting, it is possible for vertices to move back and forth between communities *ad vitam æternam* if they are evaluated concurrently for multiple iterations. The issue stems from the fact that the worker moving a vertex v would never be aware of a move occurring on a vertex

u, if said nodes always move concurrently. In this example, an algorithm that ends when no moves occurred in the previous iteration would never reach the state in which it is allowed to terminate. It is thus often necessary to introduce additional stopping conditions in parallel algorithms, such as a maximum number of iterations or an exit condition once the average improvement per iteration is under a certain threshold.

4.3.3 Previous Works

In [SM16], Staudt and Meyerhenke propose parallel algorithms based on label propagation and on the Louvain method. They further propose an ensemble preprocessing method that improves partition quality. Also based on the Louvain method, Lu, Halappanavar and Kalyanaraman [LHK15] propose a set of accessory heuristics designed to overcome some scalability issues inherent to the base algorithm. These heuristics are based on labelling and coloring. Furthermore, Moradi, Fazlali and Malazi [MFM15] propose another parallel method based on modularity maximization, in which they compute the gain in modularity obtained from moving a node from its current community to a neighbor's in parallel.

Bae et al. [BHW+13] make use of the sparsity inherent to the topological structure of a large number of graphs representing real-world systems to parallelize the Infomap algorithm. Similarly, Jin et al. [JYLY15] propose Pinfomr, a method that uses MapReduce to implement Infomap in parallel.

Prat-Pérez, Dominguez-Sal and Larriba-Pey [PPDSLP14] propose a community detection method named Scalable Community Detection (SCD), based on the maximization of a previously introduced quality function named Weighted Community Clustering (WCC) [PPDSBLP12]. In a later work by some of the same authors and others [HVPPLP15], this method is further extended further extended to heterogeneous platforms.

4.4 Proposed Solution

4.4.1 Sparsity Exploitation

As pioneered by Niu et al. [NRRW11], the sparse structure of some problems can be exploited in order to develop parallel algorithms that achieve high parallel efficiency. The sparse structure of most real-world networks can be exploited in order to parallelize algorithms for community detection in graphs [BHW+13]. While the aforementioned article studies sparsity exploitation in the single case of the Infomap algorithm, we demonstrate that graph sparsity can be exploited by a larger class of community detection algorithms.

Intuitively, the principle is simple; large complex networks are generally sparse, yet vertices that must be processed number in the tens of thousands to billions. In general, the number of processing elements available for calculation purposes is much less than that. As such, if we limit parallelism at each step to the number of processing elements - which is equivalent to considering only a very small

subset of the search space - we can achieve high parallel efficiency while keeping the probability of a collision between calculations very low.

4.4.2 Description of our Algorithms

A summary of the features of the different parallel algorithms developed in the scope of this work is available in table 4.4.1. The pseudo-code for the algorithms themselves can be found in algorithms 4.4.1 and 4.4.2. The following subsections provide a detailed explanation of the procedures as implemented.

4.4.2.1 Relaxed Parallel Louvain Method

Parallel Louvain (algorithm 4.4.1) is a parallel implementation of the Louvain algorithm [BGLL08]. Louvain is often quoted as representing the state of the art in community detection based on modularity. It is able to detect communities in very large networks in a short amount of time. However, its greedy sequential nature makes it difficult to parallelize.

The fulcrum of our algorithm is the loop over vertices in the graph that runs from line 12 to line 35. This is where most of the computation happens, and where the most benefit is obtained from parallel execution.

In the original algorithm by Blondel, Vincent and Guillaume, a vertex is first removed from its community, then the modularity gain obtained by moving it into its current (former) community, and by moving it into the community of each of its neighbors, is calculated, and subsequently the vertex is inserted into the best community available. While this method is practical in a sequential setting, it is not suitable for parallel execution as a vertice would remain 'unassigned' for a long period of time while its destination is calculated, and this could affect the results of computation effected on neighboring vertices.

In order to mitigate this effect, during each iteration we keep vertices in their community until their destination is known, at which point we update its community identifier to the destination community's. In order to speed up the calculation of modularity gains in the main loop, we keep track of the total and internal volumes of every community. These values need to be updated for both the source and destination community every time a move occurs, however, no explicit synchronization is required as we are able to take advantage of atomic updates on the x86 architecture.

We did, however, have to make a trade-off in order to use atomic updates; the reference algorithm by Blondel, Vincent and Guillaume uses `long double` types to store vertex weights and community volumes. We are not able to use this type as it uses 10 bytes of storage in memory, while we are limited to 8 bytes for atomic operations (on 64-bit architectures). The effect is a small loss of precision, `long double` being precise to 19 decimal places, while `double` types (which we use instead) are precise to 15 decimal places. However, in practice this had a marginal effect on solution quality, which was entirely dominated by other factors.

Apart from the main loop, parallelism was also introduced to other parts of the algorithm. In the initialization phase, communities and associated volumes are prepared and computed in parallel. In the second phase (coarsening) of the algorithm, the loop starting on line 43 was also parallelized. In practice, we iterate over the original graph's edges as a proxy for pairs of communities, and reduce the weights of the edges in order to form an edge on the new graph. This operation can be performed in parallel, however, synchronization is required if an edge needs to be added to the new graph, as modifying the structure of containers (`std::vector` in this case) is not a thread-safe operation in general (and in particular, in this case).

4.4.2.2 Parallel LPAm

Parallel LPAm is a parallel implementation of the modularity-specialized label propagation algorithm [BC09]. Community detection by label propagation is an extremely fast method that scales near-linearly with the number of vertices in the network. This makes it particularly interesting in the context of the study of larger networks.

As with the parallel Louvain method described in subsection 4.4.2.1, we implemented parallelism by allowing the algorithm to consider n vertices concurrently, where n is an integer smaller or equal to the number of processors available, and then allowing synchronization to allow up-to-date information to be made available to all threads. The algorithm's pseudo-code can be found in algorithm 4.4.2.

4.5 Evaluation and Results

4.5.1 Implementation and Platform

Implementation Details Our parallel algorithms were implemented in C++14 using Boost Graph version 1.62.0 [SLL01]. The OpenMP API version 4.0 (201307) [DM98] was used to support shared-memory multiprocessing. Programs were compiled using GCC 6.2.0 20161005 using optimization level `-O3` and debugging symbols were stripped from the executables. Graph visualizations were drawn using graph-tool's facilities [Tia15], and plots were drawn using matplotlib [Hun07].

Graph data structures are implemented as adjacency lists, with vertices, edges and adjacencies placed in random-access containers (`std::vector`). This is made necessary in order to be able to access the data in parallel, as sequential containers such as lists are impractical to work with in this setting. The standard memory allocator was replaced with Tcmalloc, which has more interesting performance characteristics than the standard glibc allocator for multithreaded applications.

OpenMP directives were used to implement parallel loops. Our code aims to remain lock-free, taking advantage of atomic updates made possible by the x86 architecture where necessary.

```
   Data : Graph object in adjacency list representation
   Result : Map of nodes → communities for each level, Modularity for each level
1  i ← 0;
2  while i < max_iteration ⋀ is_improved = true ∨ i = 0 do
3      is_improved ← false;
4      initialization;
5      i ← i + 1;
6      foreach Vertex v in Graph g do
7          c_v ← v;
8      end
9      phase one;
10     current_modularity ← modularity(g, c);
11     new_modularity ← current_modularity;
12     parallel foreach Vertex v in Graph g
13         current_community ← c_v;
14         best_community ← current_community;
15         c_v ← ∅;
16         gain_best ← modularity_gain(g, v, current_community);
17         foreach Neighboring vertex n of v do
18             gain_{v→c_n} ← modularity_gain(g, v, c_n);
19             if gain_{v→c_n} > gain_best then
20                 gain_best ← gain_{v→c_n};
21                 best_community ← c_n;
22             end
23             c_v ← best_community;
24             if best_community ≠ current_community then
25                 moves ← moves + 1;
26             end
27         end
28         new_modularity ← modularity(g);
29         if moves > 0 then
30             is_improved ← true;
31         end
32         if moves = 0 ⋀ new_modularity < current_modularity then
33             break;
34         end
35     end
36     Set results for level i;
37     phase two;
38     Create a new graph g′;
39     foreach Community c in Graph g do
40         Add a vertex v to g′;
41         Add a loop to v, with a weight equal to the internal weight of c;
42     end
43     parallel foreach Pair of vertices v_i, v_j
44         Add an edge e to g′, with a weight equal to the weight between c_i, c_j in g;
45     end
46     go to initialization
47 end
```

Algorithm 4.4.1 : Pseudo-code of our parallel Louvain implementation

```
   Data : Graph object in adjacency list representation, λ
   Result : Map of nodes → communities
 1 initialization;
 2 moves ← −1;
 3 i ← 0;
 4 procedure;
 5 while i ← max_iterations do
 6 |   if moves = 0 then
 7 |   |   break;
 8 |   end
 9 |   else
10 |   |   moves ← 0;
11 |   end
12 |   parallel foreach Vertex v in graph g
13 |   |   foreach Neighbor n of v do
14 |   |   |   label_score_n(v) ← label_score_n(v) + 1;
15 |   |   end
16 |   |   foreach Label l do
17 |   |   |   label_score_l ← label_score_l − λ × deg(v) × deg(l);
18 |   |   |   if label(v) = l then
19 |   |   |   |   label_score_l ← label_score_l + λ × deg(v) × deg(v);
20 |   |   |   end
21 |   |   |   if label_scores_l > max_score then
22 |   |   |   |   max_score ← l;
23 |   |   |   |   moves ← i + 1;
24 |   |   |   end
25 |   |   end
26 |   end
27 end
```

Algorithm 4.4.2 : Pseudo-code of our parallel LPAm implementation

Algorithm	Directed	Weighted	Overlapping	Hierarchical
plouvain	yes	yes	no	yes
plpam	yes	no	no	no

Table 4.4.1: Features of the algorithms described in subsection 4.4.2

Parameter	Value
Number of nodes (N)	5,000; 50,000 and 200,000
Maximum degree	$0.1 \times N$
Maximum community size	$0.1 \times N$
Average degree	20
Degree distribution exponent	3
Community size distribution exponent	1.5
Mixing coefficient μ	0.05 to 0.75, steps: 0.1

Table 4.5.1: LFR benchmark graphs parameters

Platform Results were evaluated on Amazon Web Services `m4.4xlarge` instances, which are 16-core instances running on either 2.3 GHz Intel Xeon® E5-2686 v4 (Broadwell) processors or 2.4 GHz Intel Xeon® E5-2676 v3 (Haswell) processors, with 64 gigabytes of RAM, except for the Orkut datasets which were processed on `m4.16xlarge` (same processors, 256GB RAM). The instances were running Ubuntu Linux 16.10 with kernel `4.8.0-26-generic`. No swap space was made available on the instances.

4.5.2 Datasets

In order to evaluate and benchmark the methods outlined in section 4.4, artificial (synthetic) and natural graphs will be used. The aim of this subsection is to formally introduce these datasets.

4.5.2.1 Artificial Datasets

Artificial datasets used in the scope of this work are generated using the benchmark graph algorithm described in [LFR08]. This graph generator produces graphs with a planted community structure and provides community membership information, which is of obvious use for quality evaluation purposes. The parameters used to generate our benchmark graphs are described in table 4.5.1.

The Lancichinetti-Fortunato-Radicchi (LFR) benchmark algorithm is able to generate undirected or directed, weighted or unweighted, networks with a customizable degree of overlap between communities. It can also produce hierarchical communities, in which communities are nested inside other, larger communities.

A sample visualization of an artificial network displaying community structure can be seen in figure 4.5.1, although it displays a simple planted communities model without power-law degree and community size distributions.

4.5.2.2 Real-World Datasets

Real-world relational datasets describe relationships observed in natural phenomenons. In contrast with synthetic graphs, which are generated specially for the purpose of benchmarking algo-

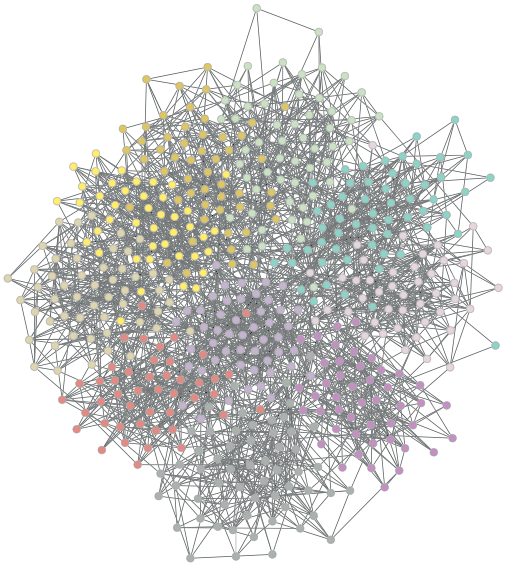

Figure 4.5.1: An artificial undirected network displaying a community structure. This network containing 500 vertices was generated using a stochastic block model with 10 blocks. The degree distribution follows a Poisson distribution with $\lambda = 10$. The network is wired such that the connection probability is 99% between vertices that share block membership, and 1% otherwise.

rithms, these graphs often do not have ground-truth community assignments available, or otherwise the ground-truth community assignments are obtained using error-prone methods and care must be taken before relying on them. Nevertheless these graphs offer more realistic testing conditions. Our network data was obtained from the Stanford Large Network Dataset Collection [LK14] and is exclusively public. A visualization of a real network can be found in figure 4.5.2.

In the scope of this work, only larger networks are considered, as the aim of our algorithms is to process larger networks in more reasonable amounts of time. The datasets studied are described below and further summarized in table 4.5.2.

LiveJournal, Orkut, YouTube, Amazon, DBLP Released by Yang and Leskovec along with [YL15], these datasets contain 230 networks with associated ground-truth data. In the scope of this article we consider the largest networks, which are well-suited for our purpose. The data was obtained by studying what the authors call *interest-based groups*, which are groups organized around a common theme that participants join explicitly.

Pokec social network This dataset by Takac and Zabovsky and released along with [TZ12], describes a very large social network comprising users of the Pokec social networking website, which is extremely popular in Slovakia. The dataset was obtained by crawling publicly accessible pages on the networking site in question.

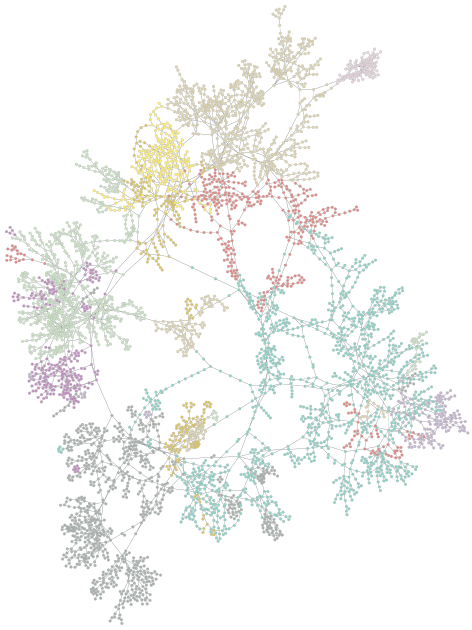

Figure 4.5.2: Community structure inferred from a dataset representing the interconnections of the Western United States power grid [WS98]. The topology of this network differs significantly from that of the artificial network shown in figure 4.5.1.

Network	$\|V\|$	$\|E\|$	$\|\Delta\|$	C	d
Pokec	1,632,803	30,622,564	32,557,458	0.1094	11
LiveJournal	3,997,962	34,681,189	177,820,130	0.2843	17
Orkut	3,072,441	117,185,083	627,584,181	0.1666	9
YouTube	1,134,890	2,987,624	3,056,386	0.0808	20
Amazon	334,863	925,872	667,129	0.3967	44
DBLP	317,080	1,049,866	2,224,385	0.6324	21

Table 4.5.2: Some properties of our real-world datasets, where $|V|$ is the number of vertices in the graph, $|E|$ the number of edges, $|\Delta|$ the number of triangles, C the average clustering coefficient, computed as $(3 \times |\Delta|)/(|V|/3)$ and d the diameter of the graph, computed as $\max_{u,v \in V} d(u, v)$.

4.5.3 Experimental Protocol

Our algorithms were evaluated according to two factors: speed of execution and quality. Results are reported for the algorithms as executed using 1, 2, 4, 8 and 16 workers. The single worker case is equivalent to a sequential implementation of the algorithm. In terms of execution speed, results are reported in terms of absolute speed and plotted accordingly.

With regards to solution quality, where reliable ground-truth data is available (artificial datasets), we report the congruence between the results of our algorithms and the reference communities using adjusted mutual information (AMI), which is defined as

$$AMI(U,V) = \frac{MI(U,V) - E[MI(U,V)]}{\max[H(U),H(V)] - E[MI(U,V)]} \tag{4.5.1}$$

where $MI(U,V)$ denotes the mutual information between clusterings U and V, $E[\cdot]$ is the expectation operator, and $H(U)$ is the entropy of clustering U.

We further report the raw objective function scores obtained by our algorithms, and compare them by scaling the number of workers from 1 to 16. This also allows us to compare against results obtained sequentially, where there is no need to cope with the difficulties we face in the parallel setting.

In every case, results reported are averaged over twenty runs in order to account for variations caused, for instance, by vertex visit orders or by delays induced by other processes running on our evaluation machine[1]. Furthermore, the results reported for synthetic datasets are averaged over ten networks generated using the same parameters, in order to account for variability induced by the generated network structure itself.

4.5.4 Results

4.5.4.1 Runtime Results

Runtime results for our real-world datasets are reported in figure 4.5.3 for the `plouvain` algorithm, and in figure 4.5.4 for the `plpam` algorithm. Speedup factors, defined as the ratio between the runtime for a parallel algorithm and its sequential counterpart, are reported in table 4.5.3.

We can observe that runtime scales similarly for all datasets under consideration. With regards to the algorithms themselves, `plpam` appears to run faster than `plouvain`, which is no surprise considering that this is also the case with the sequential algorithms. However, it also appears that `plpam` scales better than `plouvain` with regards to the number of workers. This may be because LPAm is operationally a simpler algorithm.

1. Note that the evaluation machines were dedicated to running the experiments. However, some background management processes which cannot easily be turned off were still running on the system during the experiments.

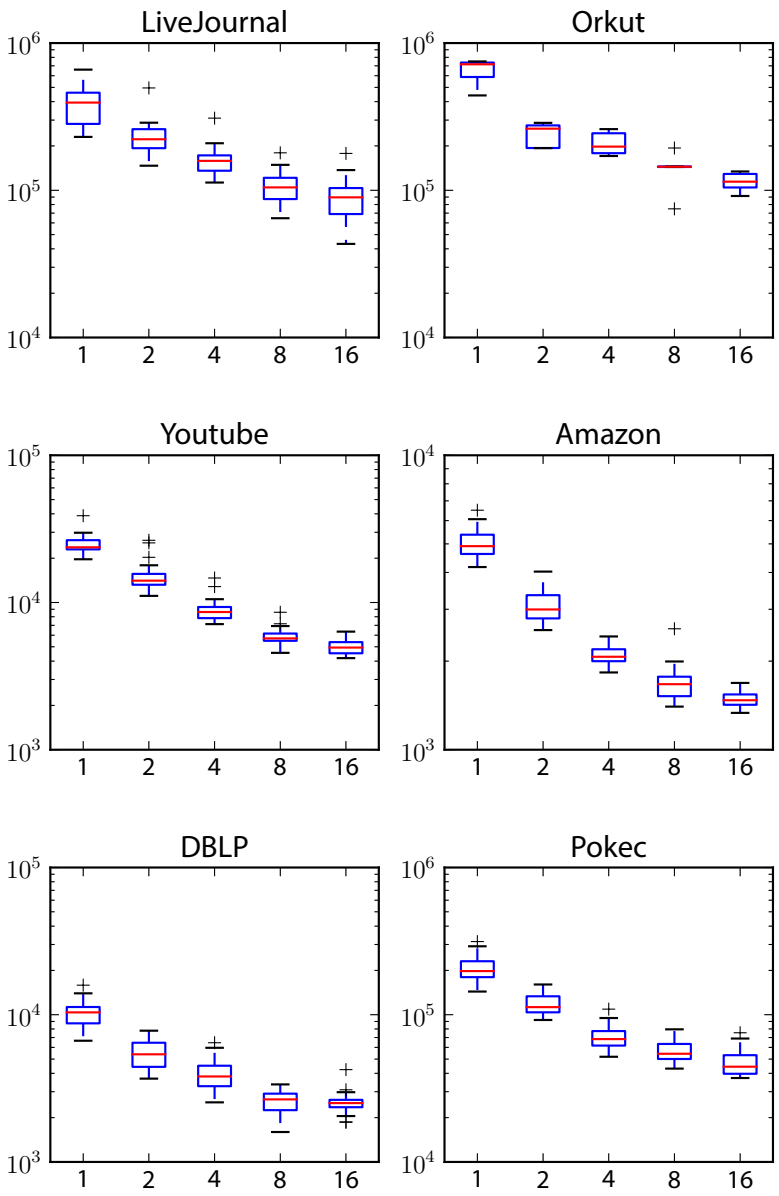

Figure 4.5.3: `plouvain` runtime benchmark (twenty runs, five for Orkut) for tests run on real-world datasets. Time is reported in milliseconds against the number of workers executing the task in parallel.

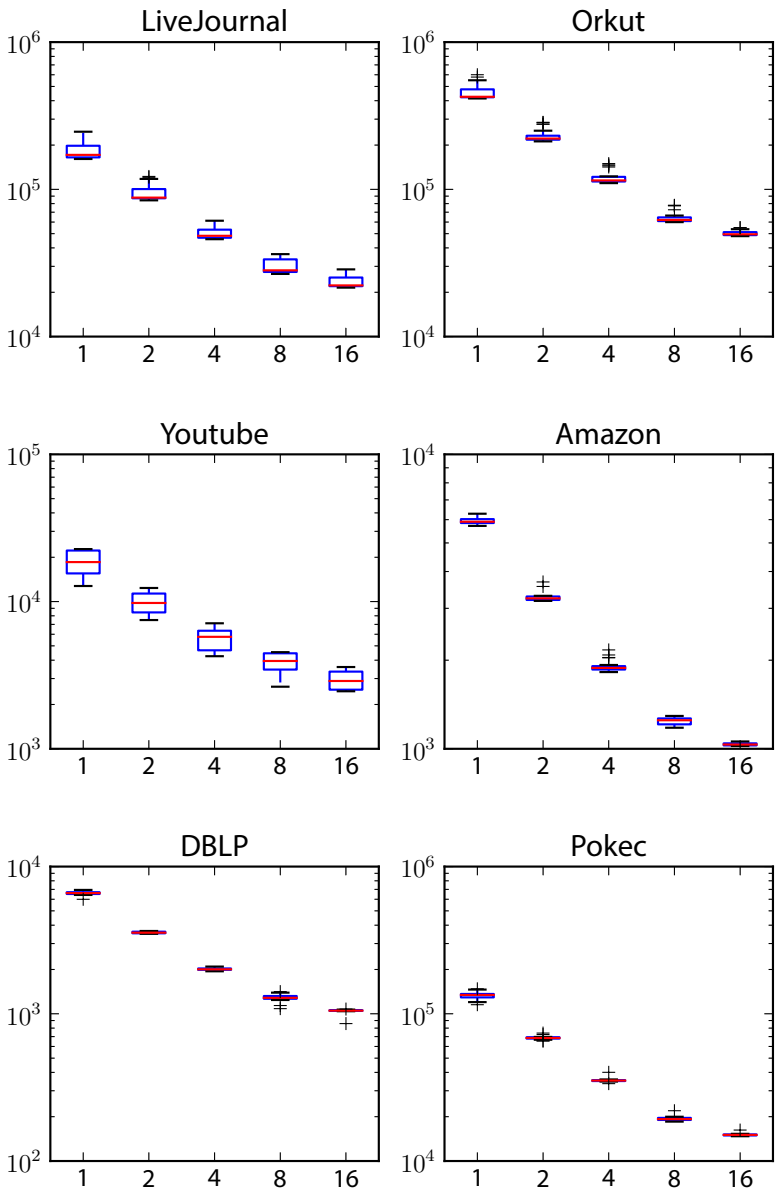

Figure 4.5.4: `plpam` runtime benchmark (20 runs) for tests run on real-world datasets. Time is reported in milliseconds against the number of workers executing the task in parallel.

Dataset	1 worker	2 workers	4 workers	8 workers	16 workers
LiveJournal	1	1.7748	2.4883	3.7682	4.4019
Orkut	1	2.7358	3.6226	4.9625	6.2673
YouTube	1	1.6850	2.7522	4.1564	4.7971
Amazon	1	1.6406	2.3748	2.9431	3.3368
DBLP	1	1.9266	2.7243	3.8993	4.1322
Pokec	1	1.7571	2.8975	3.6437	4.4612

(a) Median speedup factor for `plouvain`

Dataset	1 worker	2 workers	4 workers	8 workers	16 workers
LiveJournal	1	1.9513	3.5456	6.0841	7.7019
Orkut	1	1.9219	3.7005	6.8706	8.5893
YouTube	1	1.8954	3.2188	4.6961	6.4214
Amazon	1	1.8212	3.1382	4.7248	5.7118
DBLP	1	1.8571	3.2938	5.1453	6.2835
Pokec	1	1.9603	3.8132	6.9701	8.9218

(b) Median speedup factor for `plpam`

Table 4.5.3: Speedup factor, defined as $S = T_1/T_N$ where 1 and N are the number of workers, for benchmarks run on real-world datasets, averaged over twenty runs (five for Orkut).

4.5.4.2 Quality Measure Results

Quality results are reported in table 4.5.4. The values represent the average modularity over twenty runs for the named datasets. We note that better modularity results were obtained by the `plouvain` algorithm.

With regards to the `plpam` algorithm, increasing the number of workers does not seem to deteriorate the final quality of the solutions obtained. This result is not surprising considering that randomization is an intrinsic part of the algorithm even in the sequential case, and multiprocessor scheduling can act as a form of randomization.

This is, however, not the case with the `plouvain` algorithm, for which the best values were the results of the sequential experiment. Contrary to the `plpam` algorithm, `plouvain` is based on a greedy, sequential procedure and its heuristic steps are more sophisticated. As such, it may not benefit as much from randomization. Nevertheless, the results obtained by the multi-worker experiments are quite competitive with the results obtained by the sequential algorithm from a quality measure perspective.

4.5.4.3 Mutual Information Results

Mutual information results are reported in figure 4.5.5 for the artificial dataset experiments, which benefit from having reliable reference community assignments available for comparison. For both algorithms, we can observe that there appears to be a strong cutoff in AMI between mixing

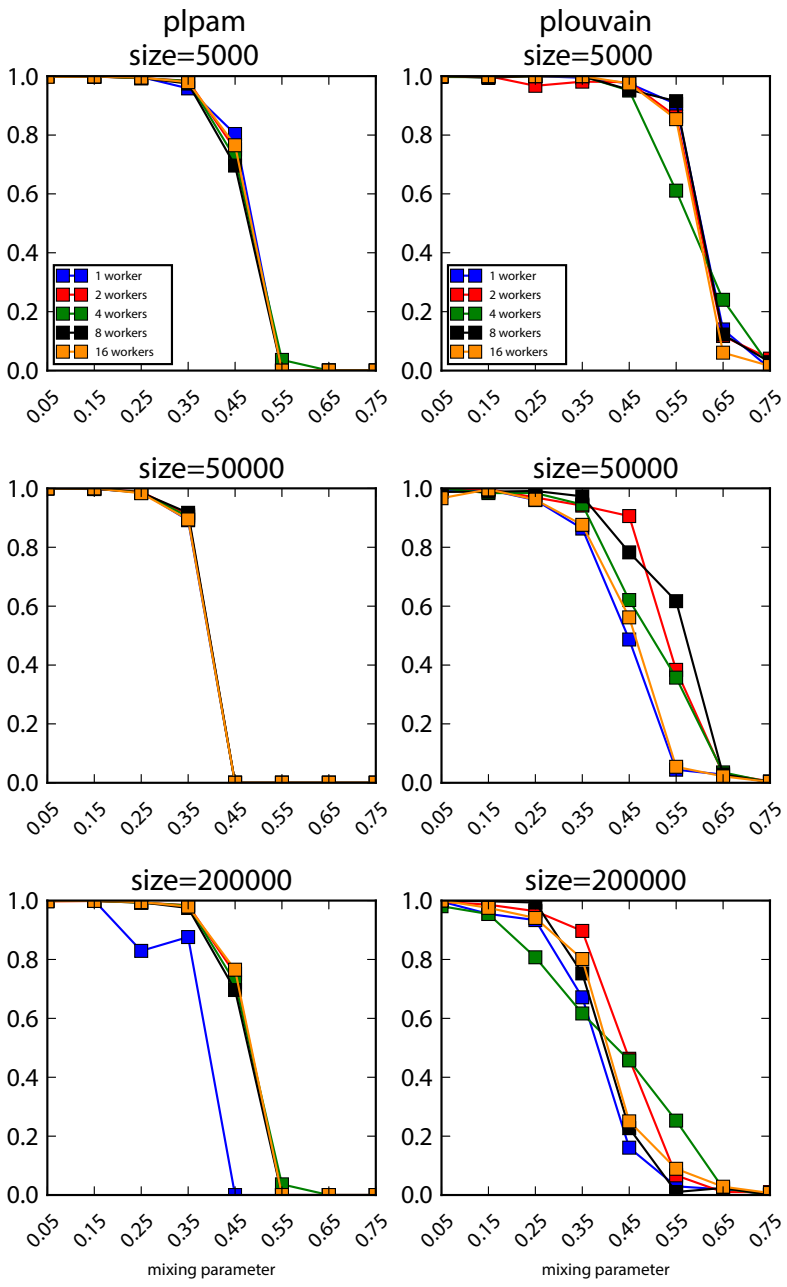

Figure 4.5.5: Adjusted mutual information benchmarks for LFR-generated datasets. Each point is averaged over twenty runs of the algorithm.

Dataset	1 worker	2 workers	4 workers	8 workers	16 workers
LiveJournal	0.6667	0.678	**0.7026**	0.6622	0.6643
Orkut	0.6059	0.6276	0.629	**0.6422**	0.6226
YouTube	**0.6345**	0.63	0.6311	0.63	0.6276
Amazon	**0.8546**	**0.8546**	0.8531	0.8501	0.8483
DBLP	**0.7639**	0.7332	0.7131	0.7059	0.6915
Pokec	**0.7038**	0.6926	0.6887	0.6889	0.6894

(a) Modularity results for `plouvain`

Dataset	1 worker	2 workers	4 workers	8 workers	16 workers
LiveJournal	0.5297	**0.5389**	0.5282	0.5332	0.5386
Orkut	0.4896	0.5004	0.5059	0.5156	**0.521**
YouTube	0.5796	0.6035	0.5928	**0.6161**	0.5872
Amazon	**0.7449**	**0.7449**	0.7448	0.7447	0.7448
DBLP	0.6289	**0.6292**	0.6277	0.6272	0.6274
Pokec	0.656	0.6831	**0.684**	0.6622	0.6799

(b) Modularity results for `plpam`

Table 4.5.4: Quality results for benchmarks run on real-world datasets, averaged over twenty runs (five for Orkut).

parameter (μ) values 0.45 and 0.55. This is as expected, since communities should not be detected reliably when $\mu > 0.5$, as this is indicative of the absence of community structure, or even of the presence of disassociative structure.

For the `plpam` algorithm in particular, we can observe that results are very similar for all experiments (1 to 16 workers). However, we also observe that multi-worker experiments achieve significantly better results than the single worker experiment, for both solution quality and stability, on the 200,000 vertices dataset. This further indicates that label propagation methods may benefit greatly from the random fluctuations introduced by parallel execution, and this should warrant further scientific exploration.

For the `plouvain` algorithm, parallelization appears to give results that compare favorably to the sequential algorithm. For larger datasets, `plouvain` appears to consistently perform better than its serial counterpart, which may indicate that it also benefits from the random fluctuations in execution steps introduced by parallel execution, but to a somewhat lesser extent.

We note that `plouvain` performs much better than `plpam` when a large amount of noise is present in the dataset. This is characterized by the mixing parameter threshold at which detection becomes unreliable, which is significantly higher for `plouvain` than `plpam`.

4.6 Conclusion

In this work we endeavored to discuss the issues that make the design of parallel community detection algorithms challenging. We further proposed algorithms that exhibit parallelism based on a number of principles and evaluated the aforementioned algorithms on synthetic and real-work networks.

Two algorithms were implemented and tested, `plouvain` and `plpam`, respectively based on the Louvain and modularity-specialized label propagation methods for community detection. We showed that these algorithms were able to obtain solutions that display quality similar to their reference sequential counterparts.

We showed that label propagation methods in particular may benefit greatly from randomization induced by the parallel execution process for the analysis of larger datasets. The results obtained on 200,000 nodes LFR graphs show that, in all cases, parallel experiments performed better than their sequential counterpart. This is an interesting result that should warrant further investigation, as it appears that these random fluctuations assist the algorithm in avoiding local minima to some degree.

Similarly, the `plouvain` algorithm performed better in parallel than its sequential counterpart for larger datasets, although the effect is less pronounced. This indicates that it also benefits from random fluctuations from a quality perspective. This may also be because this effect allows the algorithm to leave a local minima which the sequential algorithm is unable to escape.

We note that label propagation methods perform very well when contrasted with their speed and simplicity. They are able to obtain good results in short amounts of time, and scale well when the number of workers is increased. These methods may therefore be very useful in use cases where a very large relational dataset must be explored, and where trading off solution quality for speed is acceptable.

In other cases, `plouvain` appeared to detect communities more reliably, both from a quality measure and mutual information perspective. The way it scales with regards to the number of workers is, however, much less interesting than the way `plpam` scales. This may be because of the greedy iterative process which the base algorithm uses; Louvain owes its speed and effectiveness to the clever search mechanism it makes use of, and random fluctuations may cancel parts of that effect. This hypothesis should be tested in a future work.

The key takeaway from this work is that, since label propagation algorithms give high quality results in short amounts of time, and scale well when executed in parallel, they could be very useful in contexts where a large dataset needs to be analyzed, and speed is more important than solution quality. This could be useful, for instance, in recommendation systems. In problems where obtaining a very high quality solution is required, or where the dataset contains a lot of noise, `plouvain` would be a better candidate.

Chapitre 5

Conclusion

L'objectif de ce travail de recherche était l'étude de méthodes parallèles en détection de communautés dans les réseaux de grande taille. Des problématiques concrètes entourant l'exécution des algorithmes de pointe en détection de communautés ont été décrites, et des algorithmes basés sur des méthodes visant à pallier à ces dernières ont été développés et implémentés. Les algorithmes ont ensuite été évalués sur des jeux de données correspondants à des réseaux réels fréquemment utilisés pour évaluation, et à des réseaux artificiels générés avec une méthode reconnue pour son efficacité à créer des réseaux qui comportent plusieurs attributs des communautés naturelles.

Deux algorithmes ont été implémentés, soit une version parallèle de l'algorithme de Louvain, et une version parallèle de LPAm. Il est particulièrement intéressant de noter la stabilité de LPAm, qui détecte des communautés dans des temps restreints, et s'adapte très efficacement à l'exécution en parallèle. En effet, en comparaison, LPAm semble être mieux adapté à l'exécution en parallèle, mais donne des résultats en général moins bons que Louvain. Il est aussi intéressant de noter que la qualité des résultats obtenus est supérieure en parallèle que séquentiellement pour des graphes de grande taille. Ce constat pourrait indiquer que LPAm bénéficie des perturbations à l'ordre d'exécution introduites par l'exécution parallèle. Il s'agit d'une hypothèse intéressante, et ce phénomène mériterait une étude plus approfondie, afin de déterminer si cette performance accrue est réellement due à la taille du réseau, ou à d'autres facteurs latents.

Les implications de notre travail sont intéressantes pour les sciences de la gestion, en particulier en ce qui concerne l'analytique et l'intelligence d'affaires. En effet, ces domaines sont de plus en plus ammenés vers l'étude de jeux de données de très grande taille, et l'étude de ces derniers demande énormément de ressources de calcul. Nos implémentations visent à permettre l'utilisation de ces ressources plus efficacement sur un problème donné. Il est ainsi possible de réduire l'investissement en temps et en argent nécéssaire pour effectuer des analyses utiles dans un contexte industriel, scientifique ou gouvernemental afin d'informer la prise de décisions.

Le mémoire comporte toutefois nécessairement certaines limites. En particulier, seules des méthodes basées sur l'évaluation de la modularité ont été implémentées. Toutefois, la recherche théo-

rique sur la modularité a démontré que cette mesure comporte certaines lacunes qui peuvent remettre en question la qualité et la fiabilité des communautés obtenues suite à son optimisation. Il serait donc intéressant d'un point de vue pratique et académique de développer des algorithmes parallèles basés sur un principe autre dans un travail futur.

De plus, les algorithmes implémentés ont été développés afin d'être exécutés sur un seul ordinateur comportant plusieurs processeurs traditionnels avec mémoire partagée. De la recherche complémentaire pourrait être effectuée afin de développer de nouveaux algorithmes qui se parallélisent massivement, et qui seraient compatibles avec des plates-formes hétérogènes. Cela permettrait d'exploiter la puissance de grappes de serveurs et de coprocesseurs spécialisés tels que des unités de traitement graphiques à des fins générales (GPGPU). Les résultats obtenus dans le cadre de ce travail semblent toutefois indiquer qu'il s'agit d'une tâche qui pourrait s'avérer complexe.

Finalement, soulignons que les algorithmes implémentés dans le cadre de ce travail ont été développés dans l'optique d'être utiles à la communauté scientifique. Ces derniers seront par conséquent distribués au public sous une licence libre sous forme de librairie.

Bibliographie

[ACC+10] D. Aloise, S. Cafieri, G. Caporossi, P. Hansen, S. Perron, and L. Liberti. Column generation algorithms for exact modularity maximization in networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 82(4), 2010. 31

[AM11] R. Aldecoa and I. Marín. Deciphering Network Community Structure by Surprise. *PLoS ONE*, 6(9) :e24195, sep 2011. 13

[Ant71] J. M. Anthonisse. The rush in a directed graph. 1971. 16

[BA99] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(October) :509–512, 1999. 5

[Bac78] J. Backus. Can programming be liberated from the von Neumann style ?: a functional style and its algebra of programs. *Communications of the ACM*, 21(8) :613–641, aug 1978. 24

[Bar16a] B. Barney. Introduction to Parallel Computing, 2016. 19

[Bar16b] B. Barney. Message Passing Interface (MPI), 2016. 25

[Bat80] K. E. Batcher. Design of a Massively Parallel Processor. *IEEE Transactions on Computers*, C-29(9) :836–840, sep 1980. 18

[BBK+68] G. Barnes, R. Brown, M. Kato, D. Kuck, D. Slotnick, and R. Stokes. The ILLIAC IV Computer. *IEEE Transactions on Computers*, C-17(8) :746–757, aug 1968. 18

[BC01] A. Broido and K. C. Claffy. Internet topology : connectivity of IP graphs. *Scalability and Traffic Control in IP Networks*, 4526 :172–187, 2001. 4

[BC09] M. J. Barber and J. W. Clark. Detecting network communities by propagating labels under constraints. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 80(2), 2009. 32, 37

[BC15] N. D. Bhaskar and D. L. K. Chuen. Bitcoin Mining Technology. In D. L. K. Chuen, editor, *Handbook of Digital Currency*, pages 45–65. Elsevier, 2015. 23

[BDK15] V. D. Blondel, A. Decuyper, and G. Krings. A survey of results on mobile phone datasets analysis. *EPJ Data Science*, 4(1) :10, dec 2015. 4

[BGLL08] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics : Theory and Experiment*, 2008(10) :P10008, oct 2008. 13, 31, 36

[BHW+13] S.-H. Bae, D. Halperin, J. West, M. Rosvall, and B. Howe. Scalable Flow-Based Community Detection for Large-Scale Network Analysis. In *2013 IEEE 13th International Conference on Data Mining Workshops*, pages 303–310. IEEE, dec 2013. 33, 35

[BMBL09] S. P. S. Borgatti, A. Mehra, D. D. J. Brass, and G. Labianca. Network analysis in the social sciences. *Science*, 323(April) :892–5, 2009. 3

[BMS04] P. S. Bearman, J. Moody, and K. Stovel. Chains of Affection : The Structure of Adolescent Romantic and Sexual Networks. *American Journal of Sociology*, 110(1) :44–91, 2004. 4

[Buz85] B. L. Buzbee. Applications of MIMD machines. *Computer Physics Communications*, 37(1-3) :1–5, 1985. 19

[CE71] E. G. Coffman and M. J. Elphick. System Deadlocks. *Computing Surveys*, 3(2) :67–78, 1971. 27

[CGP11] M. Coscia, F. Giannotti, and D. Pedreschi. A classification for community discovery methods in complex networks. *Statistical Analysis and Data Mining*, 4(5) :512–546, oct 2011. 30

[CS07] N. Chatterjee and S. Sinha. Understanding the mind of a worm : hierarchical network structure underlying nervous system function in C. elegans, 2007. 5

[Den05] P. J. Denning. The locality principle. *Communications of the ACM*, 48(7) :19, 2005. 25

[DG04] J. Dean and S. Ghemawat. MapReduce : Simplified Data Processing on Large Clusters. *Proc. of the OSDI - Symp. on Operating Systems Design and Implementation*, pages 137–149, 2004. 23

[DGG41] A. Davis, B. B. Gardner, and M. R. Gardner. *Deep South : A Social Anthropological Study of Caste and Class*. 1941. 4

[DM98] L. Dagum and R. Menon. OpenMP : an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1) :46–55, 1998. 22, 37

[DM04] B. Drossel and A. J. McKane. Modelling food webs. In S. Bornholdt and H. G. Schuster, editors, *Handbook of Graphs and Networks*, pages 218–247. Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim, FRG, dec 2004. 5

[DP15] N. Dugue and A. Perez. Directed Louvain : maximizing modularity in directed networks. pages 0–14, 2015. 31

[DSW+04] P. De, A. Singh, T. Wong, W. Yacoub, and A. Jolly. Sexual network analysis of a gonorrhoea outbreak. *Sexually Transmitted Infections*, 80(4) :280–285, aug 2004. 4

[Dun90] R. Duncan. A Survey of Parallel Computer Architectures. *Computer*, 23(2) :5–16, 1990. 18, 19

[DWL+12] P. Du, R. Weber, P. Luszczek, S. Tomov, G. Peterson, and J. Dongarra. From CUDA to OpenCL : Towards a performance-portable solution for multi-platform GPU programming. *Parallel Computing*, 38(8) :391–407, 2012. 22

[ER59] P. Erdös and A. Rényi. On random graphs I. *Publicationes Mathematicae*, 6 :290–297, 1959. 31

[ER11] A. V. Esquivel and M. Rosvall. Compression of Flow Can Reveal Overlapping-Module Organization in Networks. *Physical Review X*, 1(2) :1–11, may 2011. 32

[Eul41] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Comentarii academiae scientiarum Petropolitanae*, 8 :128–140, 1741. 1, 28

[FB07] S. Fortunato and M. Barthelemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1) :36–41, jan 2007. 13, 31

[Fen72] T.-y. Feng. Some Characteristics of Associative/Parallel Processing. In *Proceedings of the 1972 Sagamore Computer Conference*, pages 5–16, Syracuse, 1972. 20

[FH16] S. Fortunato and D. Hric. Community detection in networks : A user guide. *Physics Reports*, 659 :1–44, nov 2016. 10, 30

[FKS86] C. Fernstrom, I. Kruzela, and B. Svensson. LUCAS associative array processor : design, programming and application studies. In *Lecture Notes in Computer Science (Book 216)*, page 326. Springer-Verlag, 1986. 18

[FLG00] G. W. Flake, S. Lawrence, and C. L. Giles. Efficient identification of Web communities. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, pages 150–160, New York City, 2000. ACM Press. 29

[Fly66] M. J. Flynn. Very High-Speed Computing Systems, 1966. 17

[Fly72] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9) :948–960, 1972. 17

[FMA+10] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello. Hardware accelerated convolutional neural networks for synthetic vision systems. *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 257–260, 2010. 23

[For10] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5) :75–174, 2010. 11, 30

[Fre77] L. C. Freeman. A Set of Measures of Centrality Based on Betweenness, 1977. 16

[FS64] T. J. Fararo and M. H. Sunshine. *A Study of a Biaised Friendship Network*. Syracuse University, 1964. 4

[FVS11] J. Fang, A. L. Varbanescu, and H. Sips. A comprehensive performance comparison of CUDA and OpenCL. In *Proceedings of the International Conference on Parallel Processing*, pages 216–225, 2011. 22

[GM78] J. Galaskiewicz and P. V. Marsden. Interorganizational resource networks : Formal patterns of overlap. *Social Science Research*, 7(2) :89–107, jun 1978. 4

[GN02] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12) :7821–7826, jun 2002. 15, 16, 32

[GPHB09] D. Göhringer, T. Perschke, M. Hübner, and J. Becker. A Taxonomy of Reconfigurable Single-/Multiprocessor Systems-on-Chip. *International Journal of Reconfigurable Computing*, 2009 :1–11, 2009. 19

[GSPA04] R. Guimerà, M. Sales-Pardo, and L. A. N. Amaral. Modularity from fluctuations in random graphs and complex networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 70(2 2), 2004. 13, 31

[Hän86] W. Händler. The Impact of Classification Schemes on Computer Architecture. In D. P. Agrawal, editor, *Advanced Computer Architecture*, pages 3–11. IEEE Computer Society Press, Los Alamitos, CA, USA, 1986. 20

[Her91] M. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1) :124–149, 1991. 27

[HK07] S. Helleringer and H.-P. Kohler. Sexual network structure and the spread of HIV in Africa : evidence from Likoma Island, Malawi. *Aids*, 21(17) :2323–2332, 2007. 4

[HLL83] P. W. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels : First steps. *Social Networks*, 5(2) :109–137, jun 1983. 11, 30

[HLM03] M. Herlihy, V. Luchangco, and M. Moir. Obstruction-free synchronization : double-ended queues as an example. *Proceedings of the 23rd International Conference on Distributed Computing Systems*, pages 522–529, 2003. 27

[HS86] W. D. Hillis and G. L. Steele. Data parallel algorithms. *Communications of the ACM*, 29(12) :1170–1183, 1986. 25

[HSN+04] A. Halaas, B. Svingen, M. Nedland, P. Saetrom, O. Snove, and O. Birkeland. A recursive MISD architecture for pattern matching. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(7) :727–734, jul 2004. 19

[Hun07] J. D. Hunter. Matplotlib : A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3) :90–95, 2007. 37

[HVPPLP15] S. Heldens, A. L. Varbanescu, A. Prat-Pérez, and J.-L. Larriba-Pey. Towards Community Detection on Heterogeneous Platforms. In S. Hunold, A. Costan, D. Giménez, A. Iosup, L. Ricci, M. E. Gómez Requena, V. Scarano, and A. L. Varbanescu, editors, *Euro-Par 2015 : Parallel Processing Workshops : Euro-Par 2015 International Workshops*, pages 209–220. Springer International Publishing, 2015. 14, 35

[JMBO01] H. Jeong, S. P. Mason, A.-L. Barabasi, and Z. N. Oltvai. Lethality and centrality in protein networks. *Nature*, 411(6833) :41–42, may 2001. 4

[JYLY15] S. Jin, P. S. Yu, S. Li, and S. Yang. A Parallel Community Structure Mining Method in Big Social Networks. *Mathematical Problems in Engineering*, 2015 :1–13, 2015. 35

[KM92] K. Kennedy and K. S. McKinley. Optimizing for parallelism and data locality. *Proceedings of the 6th international conference on Supercomputing - ICS '92*, pages 323–334, 1992. 25

[KN11] B. Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1) :016107, jan 2011. 11, 31

[KTR07] I. Kuon, R. Tessier, and J. Rose. FPGA Architecture : Survey and Challenges. *Foundations and Trends® in Electronic Design Automation*, 2(2) :135–253, 2007. 22

[LAH07] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. *ACM Transactions on the Web*, 1(1), may 2007. 29

[LFR08] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 78(4) :1–5, 2008. 40

[LGH06] S. Lämmer, B. Gehlsen, and D. Helbing. Scaling laws in the spatial structure of urban road networks. *Physica A : Statistical Mechanics and its Applications*, 363(1) :89–95, 2006. 4

[LHK15] H. Lu, M. Halappanavar, and A. Kalyanaraman. Parallel heuristics for scalable community detection. *Parallel Computing*, 47 :19–37, aug 2015. 34, 35

[LHY+11] H. Liu, X.-B. Hu, S. Yang, K. Zhang, and E. Di Paolo. Application of Complex Network Theory and Genetic Algorithm in Airline Route Networks. *Transportation Research Record : Journal of the Transportation Research Board*, 2214(-1) :50–58, 2011. 4

[LK14] J. Leskovec and A. Krevl. SNAP Datasets : Stanford Large Network Dataset Collection, jun 2014. 41

[LM10] X. Liu and T. Murata. Advanced modularity-specialized label propagation algorithm for detecting communities in networks. *Physica A : Statistical Mechanics and its Applications*, 389(7) :1493–1500, apr 2010. 33

[Mac03] M. Macedonia. The GPU enters computing's mainstream. *Computer*, 36(10) :106–108, oct 2003. 23

[Mar75] P. Mariolis. Interlocking Directorates and Control of Corporations : The Theory of Bank Control. *Social Science Quarterly (Southwestern Social Sciences Association)*, 56(3) :425–439, 1975. 4

[MFM15] E. Moradi, M. Fazlali, and H. T. Malazi. Fast parallel community detection algorithm based on modularity. In *2015 18th CSI International Symposium on Computer Architecture and Digital Systems (CADS)*, number January 2016, pages 1–4. IEEE, oct 2015. 35

[Mil67] S. Milgram. The Small World Problem. *Psychology Today*, 2 :60–67, 1967. 4

[ML12] J. J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 539–547. Curran Associates, 2012. 4

[MMG+07] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement - IMC '07*, page 29, New York, New York, USA, 2007. ACM Press. 29

[Moo06] G. E. Moore. Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff. *IEEE Solid-State Circuits Newsletter*, 20(3) :33–35, sep 2006. 21

[Mor34] J. Moreno. Part VI : Who shall survive ? In *Who shall survive ?: A new approach to the problem of human interrelations.*, pages 363–369. Nervous and Mental Disease Publishing Co., Washington D.C., 1934. 3

[MP92] H. Massalin and C. Pu. A Lock-Free Multiprocessor OS Kernel. *ACM SIGOPS Operating Systems Review*, 26(2) :8, 1992. 27

[New01] M. E. J. Newman. The Structure of Scientific Collaboration Networks. *Proceedings of the National Academy of Sciences of the United States of America*, 98(2) :404–409, 2001. 5

[New03] M. E. J. Newman. The structure and function of complex networks. *SIAM Reviews*, 45(2)(2) :167–256, 2003. 1

[New04] M. E. J. Newman. Detecting community structure in networks. *European Physical Journal B*, 38(2) :321–330, mar 2004. 13

[New06] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74(3) :036104, sep 2006. 13, 31

[New16] M. E. J. Newman. Community detection in networks : Modularity optimization and maximum likelihood are equivalent. Technical report, jun 2016. 31

[NG04] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2) :026113, feb 2004. 12, 31

[NM92] R. H. B. Netzer and B. P. Miller. What are race conditions ?: Some issues and formalizations. *ACM Letters on Programming Languages and Systems*, 1(1) :74–88, mar 1992. 26

[NRRW11] F. Niu, B. Recht, C. Re, and S. J. Wright. HOGWILD !: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. *Advances in Neural Information Processing Systems*, (1) :21, 2011. 35

[Nys14] R. Nystrom. *Game Programming Patterns*. 2014. 26

[Par83] D. Parkinson. The Distributed Array Processor (DAP). *Computer Physics Communications*, 28(4) :325–336, 1983. 18

[PCC+15] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger. A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. *IEEE Micro*, 35(3) :10–22, may 2015. 23

[PDF05] G. Palla, I. Derenyi, and I. I. Farkas. Uncovering the overlapping community structure of complex networks in nature and society - Supplementary. *Nature*, 435(7043) :814–818, jun 2005. 10, 29

[Pei14] T. P. Peixoto. Efficient Monte Carlo and greedy heuristic for the inference of stochastic block models. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 89(1) :1–8, 2014. 12, 31

[PL05] P. Pons and M. Latapy. Computing Communities in Large Networks Using Random Walks. In *Computer and Information Sciences - ISCIS 2005*, pages 284–293. Springer Berlin Heidelberg, 2005. 15, 32

[Pot85] J. L. Potter. *The Massively parallel processor*. MIT Press, 1985. 18

[PPDSBLP12] A. Prat-Pérez, D. Dominguez-Sal, J. M. Brunat, and J.-L. Larriba-Pey. Shaping communities out of triangles. *Proceedings of the 21st ACM international conference on Information and knowledge management - CIKM '12*, page 1677, 2012. 14, 35

[PPDSLP14] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey. High quality, scalable and parallel community detection for large real graphs. In *Proceedings of the 23rd international conference on World wide web - WWW '14*, pages 225–236, New York, New York, USA, 2014. ACM Press. 14, 35

[Pri65] D. J. Price de Solla. Networks of Scientific Papers. *Science*, 149(3683) :510–515, jul 1965. 5

[RAB09] M. Rosvall, D. Axelsson, and C. T. Bergstrom. The map equation. *European Physical Journal : Special Topics*, 178(1) :13–23, 2009. 32

[RAK07] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3) :036106, sep 2007. 16, 32

[RB02a] E. Ravasz and A.-L. Barabasi. Hierarchical Organization in Complex Networks. *Physical Review E*, 67(2) :026112, jun 2002. 10

[RB02b] A. Réka and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74 :47, 2002. 32

[RB06] J. Reichardt and S. Bornholdt. Statistical Mechanics of Community Detection. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 74(1), mar 2006. 31

[RB08] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4) :1118–1123, jan 2008. 15, 32

[RB11] M. Rosvall and C. T. Bergstrom. Multilevel Compression of Random Walks on Networks Reveals Hierarchical Organization in Large Integrated Systems. *PLoS ONE*, 6(4) :e18209, apr 2011. 32

[RCC+04] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, 101(9) :2658–2663, mar 2004. 9, 29

[RH61] A. Rapoport and W. J. Horvath. A study of a large sociogram. *Behavioral Science*, 6(4) :279–291, 1961. 4

[SCBR14] S. Sobolevsky, R. Campari, A. Belyi, and C. Ratti. General optimization technique for high-quality community detection in complex networks. *Physical Review E*, 90(1) :012811, jul 2014. 14

[Seg92] P. O. Seglen. The skewness of science. *Journal of the American Society for Information Science*, 43(9) :628–638, oct 1992. 5

[SGS10] J. E. Stone, D. Gohara, and G. Shi. OpenCL : A parallel programming standard for heterogeneous computing systems. *Computing in Science and Engineering*, 12(3) :66–72, 2010. 22

[Sha48] C. E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27(3) :379–423, jul 1948. 15

[Shi12] R. Shields. Cultural Topology : The Seven Bridges of Konigsburg, 1736. *Theory, Culture & Society*, 29(4-5) :43–57, jul 2012. 28

[SLJRT10] P. P. A. Staniczenko, O. T. Lewis, N. S. Jones, and F. Reed-Tsochas. Structural dynamics and robustness of food webs. *Ecology Letters*, 13(7) :891–899, 2010. 5

[SLL01] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library : User Guide and Reference Manual*. Addison-Wesley Professional, first edition, 2001. 37

[SM16] C. L. Staudt and H. Meyerhenke. Engineering Parallel Algorithms for Community Detection in Massive Networks. *IEEE Transactions on Parallel and Distributed Systems*, 27(1) :171–184, jan 2016. 35

[Sve92] B. Svensson. SIMD Processor Array Architectures. In H. W. Lawson, editor, *Parallel Processing in Industrial Real-Time Applications*. Prentice-Hall, 1992. 18

[Tay13] M. B. Taylor. Bitcoin and the age of Bespoke Silicon. In *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, CASES 2013*, 2013. 23

[Tia15] P. P. Tiago. The graph-tool python library. sep 2015. 37

[TLT07] K. L. Tew, X.-L. Li, and S.-H. Tan. Functional centrality : detecting lethality of proteins in protein interaction networks. In *Genome informatics. International Conference on Genome Informatics*, volume 19, pages 166–77, 2007. 4

[TM69] J. Travers and S. Milgram. An Experimental Study of the Small World Problem. *Sociometry*, 32(4) :425, dec 1969. 4

[Tru94] R. J. Trudeau. *Introduction to Graph Theory*. Dover Books on Mathematics, 2nd edition, 1994. 5

[TZ12] L. Takac and M. Zabovsky. Data Analysis in Public Social Networks. In *International Scientific Conference & International Workshop Present Day Trends of Innovations*, Lomza, 2012. 41

[VG93] J. Von Neumann and M. D. Godfrey. First Draft of a Report on the EDVAC. *IEEE Annals of the History of Computing*, 15(4) :27–75, 1993. 21

[War63] J. H. Ward. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301) :236–244, mar 1963. 15, 32

[WBB76] H. C. White, S. A. Boorman, and R. L. Breiger. Social Structure from Multiple Networks . I . Blockmodels of Roles and Positions. *American Journal of Sociology*, 81(4) :730–780, 1976. 30

[Whi15] T. White. *Hadoop : The definitive guide*, volume 4th. O'Reilly Media, 4th edition, 2015. 23

[WS98] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684) :440–442, jun 1998. 4, 42

[WSTA12] S. Wienke, P. Springer, C. Terboven, and D. An Mey. OpenACC - First experiences with real-world applications. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7484 LNCS, pages 859–870, 2012. 22

[WSTB86] J. G. White, E. Southgate, J. N. Thomson, and S. Brenner. The structure of the nervous system of the nematode Caenorhabditis elegans. *Philosophical Transactions of the Royal Society of London*, 314(1165) :1–340, 1986. 5

[XCC13] R. Xu, S. Chandrasekaran, and B. Chapman. Exploring Programming Multi-GPUs Using OpenMP and OpenACC-Based Hybrid Model. In *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, pages 1169–1176. IEEE, may 2013. 22

[YL15] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1) :181–213, jan 2015. 4, 41

[Zac77] W. W. Zachary. An Information Flow Model for Conflict and Fission in Small Groups. *Journal of Anthropological Research*, 33(4) :452–473, 1977. 29