Algorithms and Software for the Analysis of Large Complex Networks

Thesis · January 2016

DOI: 10.5445/IR/1000056470

1 author:

Christian L. Staudt
https://clstaudt.me/
35 PUBLICATIONS **978** CITATIONS

SEE PROFILE

ALGORITHMS AND SOFTWARE FOR THE ANALYSIS OF LARGE COMPLEX NETWORKS

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

CHRISTIAN LORENZ STAUDT

Tag der mündlichen Prüfung:	21. Juni 2016
erster Gutachter:	Juniorprof. Dr. Henning Meyerhenke
zweiter Gutachter:	Prof. Dr. Ulrik Brandes

ACKNOWLEDGEMENTS

First of all, I would like to thank Henning Meyerhenke for giving me the opportunity to work in his group and advising me during the past four years. I also thank Ulrik Brandes for taking an interest in my research and agreeing to act as second referee. A big thank you goes to Sasha Gutfraind and Ilya Safro for hosting and advising me during my stay in Chicago and Clemson. I would like to show my gratitude to my colleagues, in particular Elisabetta Bergamini, Roland Glantz, Michael Hamann and Moritz von Looz, who were never too busy to provide support, feedback, critical discussion and encouragement, not only in research questions. Several research projects presented in this thesis would also not have been possible without the contributions of talented students who I advised during their thesis work, research assistance jobs or internships. Special thanks in this regard is due to Maximilian Vogel, Gerd Lindner and Aleksejs Sazonovs. My thanks also go to Pratistha Bhattarai, Mark Erb, Kolja Esders, Jannis Koch, Yassine Marrakchi and Arie Slobbe for their contributions. Many more people (listed in the respective chapter) contributed to NetworKit, helping to make it a vibrant open-source project. Elisabetta Bergamini, Michael Hamann, Moritz von Looz, Holger Mühlsteph and Oliver Schneider kindly agreed to proofread parts of this thesis. Last but not least I thank Bernd Giesinger and Ralf Kölmel for technical support, and Lilian Beckert and Simone Meinhart for their help with office work.

Work presented in this thesis was supported by the project *Parallel Analysis of Dynamic Networks – Algorithm Engineering of Efficient Combinatorial and Numerical Methods*, funded by the Ministry of Science, Research and the Arts Baden-Württemberg, and by the German Research Foundation (Deutsche Forschungsgemeinschaft) under grants ME 3619/3-1 and WA 654/22-1 within the Priority Programme 1736 *Algorithms for Big Data*.

CONTENTS

SUMMARY

This thesis is composed of five parts, one introductory part followed by four parts presenting contributions in different subtopics. The work presented intersects three main areas, namely graph algorithmics, network science and applied software engineering. Graph algorithmics is concerned with developing and analyzing algorithms that operate on graphs, mathematical objects defined as a set of nodes and a set of edges connecting them, for which a rich mathematical theory exists. Network science is an emerging field of study and refers essentially to a form of data analysis (i.e. processes concerned with inspecting, transforming, and modeling data with the goal of discovering useful information) that focuses on network data, i.e. data on relationships between entities. Analysis methods of this kind become more and more relevant as many branches of science and technology – seeking to understand complex systems – shift focus from the separate parts that make up the system to the relationships and interdependencies between them. Attempts to understand social phenomena with the help of relational data collected by social media and online social networking services are a good example for this, but possible applications of network science methodology have a much wider scope. The resulting network data has a natural representation in the form of graphs, and motivates the design and evaluation of graph algorithmic methods to analyze it. Each computational method discussed in this thesis relates to at least one of the main tasks of data analysis: to extract structural features from network data, such as methods for community detection (Part III); or to transform network data, such as methods to sparsify a network and reduce its size while keeping essential properties (Part IV); or to realistically model networks, for instance through generative models (Part V). Nowadays the growing volume of relevant network data increases the demand for highly scalable methods. Evaluating and achieving scalability is therefore a major aspect throughout this thesis, aiming for methods that are in practice applicable to large networks, or a large number of networks. Techniques employed to achieve scalability include the introduction of parallelism, the development of heuristics for computationally expensive problems, or the use of efficient data structures. Furthermore, my focus is not just on developing and studying graph algorithmic methods for network science, but but also to robustly implement them and make them readily available as software tools, accessible to users within and outside of computer science. This is a main goal of the NetworKit project (Part II).

I briefly summarize the content and contributions of each part in the following. Parts of this thesis have previously appeared as conference or journal publications. For more bibliographical information on the respective publications, refer to the section *Publications* in the Appendix.

Part I: Introduction

The algorithmic methods and software tools that make up the contributions of this thesis find their main applications in the emerging field of network science. Chapter 1 establishes a context by first discussing examples of network studies, then continues by reviewing formal definitions of network data and its different mathematical representations. Based on these definitions, Chapter 2 describes a catalog of basic network analysis methods that are often referred to and employed in the course of this thesis. Chapter

3 clarifies general objectives of the research presented in this thesis in more detail and discusses methodological questions that apply throughout.

Part II: NetworKit: A Tool Suite for the Analysis of Large Complex Networks

NetworKit is an open-source network analysis software package to which I have contributed as initiator, project maintainer, software architect and developer of various components. NetworKit provides both established and novel analysis algorithms, including those presented in this thesis. It serves as the software framework on which implementations for all algorithmic contributions presented in the following have been built, as well as a suite of tools with which the majority of network analysis results presented have been generated. Apart from serving as a code base for algorithm engineering work, NetworKit is designed to support users from various fields in the exploratory analysis of large network data sets. Chapter 4 discusses design goals and design principles as well as algorithm and implementation patterns employed to develop scalable solutions. Chapter 5 gives an overview of the functionality that NetworKit provides, and gives examples of its applicability as an algorithm library and a tool for exploratory data analysis in different scenarios. NetworKit interface makes it easy to generate structural profiles of networks, supporting pattern discovery and hypothesis formation.
Chapter 6 focuses on evaluating NetworKit's performance in comparison with competing solutions for network analysis. These include existing software packages with similar target use cases, target platforms and design choices. Aspects of the comparison include functionality, user interaction, and supported platforms. For the most relevant competitors, it is shown how NetworKit compares in a performance benchmark on typical network analysis tasks. In comparison, NetworKit provides the most scalable solutions for a variety of analysis tasks in terms of running time and memory footprint. We also show through an experimental study that distributed computing solutions, which are often prominently considered for large-scale graph analysis, come with a significant overhead, suggesting the use of shared-memory parallel solutions like NetworKit as long as the graph fits into the main memory.

The paper *NetworKit: A Tool Suite for Large-scale Complex Network Analysis* (coauthored with Aleksejs Sazonovs and Henning Meyerhenke) describes the main aspects of NetworKit and is currently in the revision process for the journal *Network Science*. The study comparing NetworKit with distributed graph computing frameworks has been published as *Complex Network Analysis on Distributed Systems: An Empirical Comparison* (coauthored with Jannis Koch, Maximilian Vogel and Henning Meyerhenke) and presented at *International Symposium on Foundations and Applications of Big Data Analytics (FAB 2015)*.

Part III: Community Detection in Complex Networks

Community detection in networks is the task of dividing a network into subgraphs which are internally densely and externally sparsely connected. It is a fundamental analysis method that reveals the modular composition of a network. Chapter 7 discusses general concepts, terminology, formalisms (e.g. the target function *modularity*) and methodological questions for community detection. Chapter 8 focuses on *global community detection* in large-scale networks, i.e. finding all communities of a given graph. The problem

has been extensively studied before, yielding several heuristics, but few of them geared towards parallelism. Our algorithm engineering efforts yield two effective parallel heuristics for finding communities with high modularity: PLP, a parallel label-propagation scheme, and PLM, a parallelization of the Louvain method. PLM is extended by an optional refinement phase that yields small improvements in modularity at reasonable computational cost. In extensive comparative experiments with state-of-the-art algorithms and implementations, PLP and PLM are shown to be on the Pareto front with respect to the modularity/running time tradeoff. The result are scalable and robust implementations in NetworKit, processing close to a billion edges per minute on typical hardware. Chapter 9 is concerned with the subproblem of *selective community detection*: Given a set of seed nodes, selective community detection methods find only the communities to which these seed nodes belong. Numerous approaches have been proposed in the literature, but their effectiveness has been less than well understood for lack of comparative work. We implement and compare several methods, including an approach that is novel in this context. We thereby close a gap in terms of experimental evaluation and contribute to the clarification of the state of the art, but also conclude that the considered algorithms either lack effectiveness with respect to retrieving ground-truth communities or suffer from parameter sensitivity that limits their practicality for explorative network analysis. In most practical applications, fast and parameter-free global methods (Chapter 8) are likely superior.

Results have been published as *Engineering High-Performance Community Detection Heuristics for Massive Graphs* (coauthored with Henning Meyerhenke) at the *International Conference on Parallel Processing (ICPP 2013)*, and an extended version as *Engineering Parallel Algorithms for Community Detection in Massive Networks* in the *IEEE Transactions on Parallel and Distributed Systems*. The chapter on selective community detection is based on joint work with Yassine Marrakchi and Henning Meyerhenke, and has been presented as *Detecting Communities around Seed Nodes in Complex Networks* at the *First International Workshop on High Performance Big-Graph Data Management, Analysis and Mining* and published in the proceedings of the *IEEE International Conference on Big Data (IEEE BigData 2014)*.

Part IV: Edge Centrality Measures for Network Sparsification

Part IV contains a comparative study of edge centrality measures for the purpose of sparsifying complex networks. We start from the idea that not all edges are equally important for the structural properties of a network. By quantifying their importance (through various centrality measures) and filtering the network, sparsified versions of the network can be derived that preserve many relevant properties (such as degree distribution, diameter, node centralities etc.) containing only 20% of edges. Existing and novel methods are implemented in NetworKit and compared for effectiveness. Chapter 10 explains how we conceptualize sparsification as computing edge centrality scores followed by filtering edges by these scores, and describes several such centrality measures. These methods are then evaluated on a large set of social networks, clarifying their performance and showing that the LocalDegree measure we propose efficiently preserves a wide range of properties (Chapter 11).

Part IV is based on joint work with Gerd Lindner, Michael Hamann, Henning Meyerhenke and Dorothea Wagner. Results appeared as *Structure-Preserving Sparsification of Social Networks* in the proceedings of the *IEEE/ACM International Conference on Ad-*

vances in Social Networks Analysis and Mining (ASONAM 2015). An extended version has recently been accepted for publication in the journal *Social Network Analysis and Mining*.

Part V: Generative Models for Realistic Synthetic Networks

In experimental algorithmics, random synthetic instances are a standard tool, but their use should be backed by evidence that results obtained on synthetic instances are representative for behavior on real data. For algorithms operating on complex networks, where much depends on the structure of input networks, the problem is especially relevant and especially complicated. Accordingly, Part V focused on generative models for realistic synthetic graphs. Related work has proposed a variety of models, some of them with claims of comprehensive realism, i.e. matching patterns commonly observed in real complex networks. We approach a definition of realism from two different angles, as structure replication and running time replication.

Beyond the goal of creating replicas of the same size, in this work we specifically target the use case of producing a scaled replica of a given original network, which has so far not received much attention in the literature. We propose suitable fitting schemes which parametrize the considered models in order to generate a scaled-up version of the input graph. As the most promising approach for both goals, we propose the LFR+ generator, a modification of the LFR model for community detection benchmarks. We harness the true flexibility of LFR's algorithmic methods to increase the realism of the replication. Our fast implementation in NetworKit generates graphs according to the plain LFR and extended LFR+ models and does so significantly faster than the reference implementation, also by introducing parallelism. LFR+ improves on its predecessor LFR in terms of flexibility, realism, and efficiency of implementation. Specifically, LFR+ is generally more realistic than the RMAT, BTER and Hyperbolic Unit Disk Graph models, all of which have been proposed as realistic to the point of yielding substitutes of real network data. We show that our design yields a scalable and effective tool for replicating a given network – and possibly scale it by orders of magnitude – while closely preserving important properties on the micro- and macro level. This yields realistic test data for the engineering of computational methods on networks where suitable real data is not available.

Part V is based on joint work with Sasha Gutfraind, Ilya Safro, Henning Meyerhenke and Michael Hamann, which is unpublished at the time of completion of this thesis.

ZUSAMMENFASSUNG

Diese Dissertation besteht aus fünf Teilen, einem einleitenden Teil gefolgt von vier Teilen, die jeweils Beiträge zu verschiedenen Teilthemen präsentieren. Die hier dargestellte Arbeit befindet sich in der Schnittmenge von drei Fachgebieten, nämlich Graphenalgorithmik, *Network Science* und angewandte Softwaretechnik. Die Graphenalgorithmik befasst sich mit der Entwicklung und Analyse von Algorithmen zur Verarbeitung von Graphen, mathematischen Objekten bestehend aus einer Menge von Knoten und einer Menge von diese verbindenden Kanten, für die eine reichhaltige mathematische Theorie existiert. *Network Science* – die sich im Entstehen befindende Wissenschaft der Netzwerke – bezeichnet im Wesentlichen eine Form der Datenanalyse (d.h. Prozesse zur Untersuchung, Transformation und Modellierung von Daten mit dem Ziel, nützliche Informationen zu entdecken), die sich auf Netzwerkdaten fokussiert, d.h. Daten zu den Beziehungen zwischen Entitäten. Analysemethoden dieser Art werden in dem Maße zunehmend relevant, wie sich aktuell in vielen Bereichen von Wissenschaft und Technik, die komplexe Systeme zu verstehen versuchen, das Blickfeld verlagert – von den separaten Teilen eines Systems hin zu den Beziehungen und Interdependenzen zwischen ihnen. Der Versuch, soziale Phänomene mithilfe der von Social Media- und Social Networking-Diensten im Internet gesammelten relationalen Daten zu verstehen, ist ein treffendes Beispiel dafür, die Bandbreite an möglichen Anwendungen dieser Methodik ist jedoch viel größer. Die im Zuge dessen gesammelten Netzwerkdaten lassen sich natürlicherweise in Form von Graphen repräsentieren. Dies motiviert die Entwicklung und Evaluierung von Graphenalgorithmen zu ihrer Analyse. Jeder der in dieser Dissertation besprochenen Methoden steht in Verbindung mit mindestens einer der Hauptaufgabenstellungen der Datenanalyse: es geht darum, strukturelle Eigenschaften eines Netzwerks zu bestimmen, beispielsweise mit den Methoden zur Community Detection (Teil III); oder Netzwerkdaten zu transformieren, so beispielsweise mit Methoden, die Netzwerke ausdünnen und die Datenmenge verringern, während wichtige Struktureigenschaften erhalten bleiben (Teil IV); oder Netzwerke realistisch zu modellieren, beispielsweise mithilfe von generativen Modellen (Teil V). Heutzutage führt die wachsende Menge an relevanten netzwerkförmigen Daten zu einem Bedarf an hochskalierbaren Methoden. Skalierbarkeit zu bewerten und zu erreichen ist deshalb ein wesentlicher Aspekt in allen Teilen dieser Arbeit. Wir zielen auf Methoden ab, die in der Praxis auf große Netzwerke oder eine große Anzahl an Netzwerken effizient anwendbar sind. Unter den Techniken, die wir anwenden, um Skalierbarkeit zu erreichen, sind beispielsweise die Einführung von Parallelverarbeitung, die Entwicklung von Heuristiken für rechenintensive Probleme, oder die Verwendung effizienter Datenstrukturen. Darüber hinaus liegt der Schwerpunkt nicht nur bei der Entwicklung und Analyse graphenalgorithmischer Methoden, sondern auch darauf, sie robust zu implementieren und sie in Form von Softwarewerkzeugen griffbereit für Anwender innerhalb und außerhalb der Informatik zur Verfügung zu stellen. Dies ist ein Hauptziel des Projektes NetworKit.

Im Folgenden fasse ich kurz den Inhalt und die Beiträge jedes Teils meiner Dissertation zusammen. Teile dieser Arbeit sind zuvor als Konferenzpapiere oder Zeitschriftenartikel erschienen. Für weitere bibliographische Informationen verweise ich auf den Abschnitt *Publications* im Anhang.

Teil I: Einleitung

Die algorithmischen Methoden und Softwarewerkzeuge, die die Beiträge dieser Arbeit darstellen, finden ihre Anwendung im jungen Wissenschaftsfeld *Network Science*. Kapitel 1 baut einen Kontext auf, indem zuerst Beispiele für Netzwerkstudien diskutiert werden, dann eine formale Defintion von Netzwerkdaten und ihren verschiedenen mathematischen Repräsentationen. Basierend auf diesen Definitionen beschreibt Kapitel 2 einen Katalog von grundlegenden Netzwerkanalysemethoden, die im Zuge der Dissertation häufig referenziert und angewendet werden. Kapitel 3 erläutert allgemeine Zielsetzungen der hier präsentierten Forschung und bespricht methodologische Fragestellungen, die für alle Teile der Arbeit gelten.

Teil II: NetworKit: Ein Werkzeugkasten für die Analyse großer komplexer Netzwerke

NetworKit ist ein Open-Source Softwarepaket zur Netzwerkanalyse, zu dem ich als Initiator, Maintainer, Softwarearchitekt und Entwickler verschiedener Komponenten beigetragen habe. NetworKit stellt sowohl etablierte als auch innovative Netzwerkanalysealgorithmen zur Verfügung, inklusive der in dieser Arbeit präsentierten. Es dient als Softwareframework, auf dem alle Implementierungen der im folgenden dargestellten algorithmischen Beiträge basieren, sowie als Palette von Werkzeugen, mit denen der Großteil der dargestellten Netzwerkanalyseergebnisse generiert wurde. Abgesehen von seiner Funktion als Codebasis für Algorithm Engineering zielt NetworKit darauf ab, Anwender aus verschiedenen Fachgebieten bei der explorativen Analyse großer Netzwerke zu unterstützen. Kapitel 4 bespricht Ziele und Grundlagen für das Design von NetworKit, sowie wiederkehrende Muster bei Algorithmenentwurf und Implementierung, die angewandt wurden, um skalierbare Lösungen zu entwickeln. Kapitel 5 liefert eine Übersicht der bereitgestellten Funktionalität, sowie Anwendungsbeispiele als Algorithmenbibliothek und als Werkzeug zur explorativen Datenanalyse in verschiedenen Szenarien. Die Benutzerschnittstelle von NetworKit ermöglicht es dem Nutzer, Strukturprofile von Netzwerken zu erstellen und dabei Muster in den Daten zu erkennen und darüber Hypothesen zu bilden. Kapitel 6 evaluiert die Performance von NetworKit im Vergleich mit konkurrierenden Softwarelösungen zur Netzwerkanalyse. Diese beinhalten Softwarepakete mit ähnlichen Anwendungsszenarien, Zielplattformen und Designentscheidungen. Verglichen Aspekte beinhalten Funktionalität, Nutzerinteraktion und unterstützte Plattformen. Für die relevantesten Konkurrenten zeigt ein Benchmark, wie NetworKit bei typischen Analyseaufgaben im Vergleich abschneidet. Dabei stellt sich heraus, dass NetworKit in Punkto Laufzeit und Speicherverbrauch die skalierbarsten Lösungen für eine Reihe von Analyseaufgaben liefert. Wir zeigen ebenfalls im Rahmen einer experimentellen Studie, dass Lösungen zum verteilten parallelen Rechnen, die häufig an erster Stelle für die Analyse großer Graphen in Betracht gezogen werden, mit einem erheblichen Overhead behaftet sind, sodass es angeraten ist, Shared Memory-Lösungen wie NetworKit zu verwenden, solange der Graph in den Hauptspeicher passt.

Das Papier *NetworKit: A Tool Suite for Large-scale Complex Network Analysis* (in Koautorenschaft mit Aleksejs Sazonovs und Henning Meyerhenke) beschreibt die wichtigsten Aspekte von NetworKit und befindet sich zur Zeit im Begutachtungsprozess für die Zeitschrift *Network Science*. Die Studie, die NetworKit mit Frameworks für verteilte Systeme vergleicht, erschien als *Complex Network Analysis on Distributed Systems: An Empirical Comparison* (in Koautorenschaft mit Jannis Koch, Maximilian Vogel and Henning

Meyerhenke) und wurde im Rahmen des *International Symposium on Foundations and Applications of Big Data Analytics (FAB 2015)* präsentiert.

Teil III: Community Detection in komplexen Netzwerken

Community Detection bezeichnet die Aufgabenstellung, ein Netzwerk in Subgraphen zu zerlegen, die intern dicht und nach außen hin dünn verbunden sind. Das ist eine grundlegende Analysemethode, die den modularen Aufbau eines Netzwerks offenlegt. Kapitel 7 bespricht allgemeine Konzepte, Terminologie, Formalismen (z.B. die Zielfunktion *Modularity*) and methodologische Fragestellungen zum Thema Community Detection. Im Fokus von Kapitel 8 steht *globale Community Detection* in großen Netzwerken, d.h. die Ermittlung aller Communities eines gegebenen Graphen. Das Problemfeld wurde bereits ausgiebig bearbeitet, woraus sich einige Heuristiken ergaben, doch nur wenige davon sind auf Parallelverarbeitung ausgerichtet. Unser Algorithm Engineering ergibt zwei effektive parallele Heuristiken, um Communities mit hoher Modularity zu finden: PLP, ein Verfahren, was auf der parallelen Ausbreitung von Knotenlabels basiert, und PLM, eine Parallelisierung der sog. Louvain-Methode. PLM wird erweitert durch eine optionale Verfeinerungsphase, die zu einer leichten Verbesserung der Modularity zum Preis eines geringfügig erhöhten Rechenaufwands führt. In einem umfassenden experimentellen Vergleich mit dem Stand der Technik wird gezeigt, dass PLP und PLM bezüglich des Kompromisses zwischen Modularity und Laufzeit auf der Pareto-Front liegen. Das Ergebnis dieser Arbeit sind skalierbare und robuste Implementierungen in NetworKit, die auf typischer Hardware Verarbeitungsraten von nahezu einer Milliarde Kanten pro Minute erzielen. Kapitel 9 befasst sich mit dem Teilproblem *selektiver Community Detection*: Selektive Community Detection-Methoden finden zu einer gegebenen Menge von initialen Knoten nur die Communities, zu denen diese initialen Knoten gehören. Zahlreiche Verfahren wurden in der Literatur vorgestellt, doch ihre Effektivität ist unzureichend geklärt aufgrund des Mangels an vergleichender Arbeit. Wir implementieren und vergleichen mehrere Methoden, darunter eine in diesem Kontext neuartige Methode. Dadurch schließen wir eine Lücke in der experimentellen Evaluation und tragen zur Klärung des Stands der Technik bei, kommen aber auch zu dem Schluss, dass die betrachteten Algorithmen entweder unzureichend effektiv sind oder höchst empfindlich auf Parameteränderungen reagieren, was ihren praktischen Nutzen für explorative Netzwerkanalyse begrenzt. Für die meisten Anwendungen sind schnelle, parameterfreie globale Methoden (Kapitel 8) mit hoher Wahrscheinlichkeit überlegen.

Ergebnisse wurden als *Engineering High-Performance Community Detection Heuristics for Massive Graphs* (in Koautorenschaft mit Henning Meyerhenke) auf der *International Conference on Parallel Processing (ICPP 2013)* und in erweiterter Fassung als *Engineering Parallel Algorithms for Community Detection in Massive Networks* in der Zeitschrift *IEEE Transactions on Parallel and Distributed Systems* publiziert. Das Kapitel zur selektiven Community Detection basiert auf gemeinsamer Arbeit mit Yassine Marrakchi und Henning Meyerhenke, und wurde als *Detecting Communities around Seed Nodes in Complex Networks* im *First International Workshop on High Performance Big-Graph Data Management, Analysis and Mining* präsentiert und im Tagungsband der *IEEE International Conference on Big Data (IEEE BigData 2014)* veröffentlicht.

Teil IV: Kantenzentralitätsmaße zur Ausdünnung von Netzwerken

Teil IV enthält eine vergleichende Studie von Kantenzentralitätsmaßen zum Zweck der Ausdünnung komplexer Netzwerke. Wir gehen von der Idee aus, dass nicht alle Kanten gleichermaßen wichtig sind für die strukturellen Eigenschaften eines Netzwerks. Indem wir ihre Wichtigkeit mittels verschiedener Zentralitätsmaße quantifizieren und anhand dessen das Netzwerk filtern, erhalten wir ausgedünnte Versionen des Netzwerks, die nur noch bis zu 20% der Kanten enthalten und dennoch viele relevante Eigenschaften (z.B. Gradverteilung, Durchmesser, Knotenzentralitäten etc.) erhalten. Existierende und neuartige Methoden wurden in NetworKit implementiert und bezüglich ihrer Effektivität verglichen. Kapitel 10 erklärt, wie wir Ausdünnung als die Berechnung von Kantenzentralitäten gefolgt von einem Filterungsschritt auffassen, und beschreibt einige solcher Zentralitätsmaße. Diese Methoden werden dann auf einer großen Menge an sozialen Netzwerken evaluiert, um ihre Leistungsfähigkeit zu klären. Wir zeigen, dass das von uns vorgeschlagene Maß LocalDegree effizient ein breites Spektrum an Eigenschaften erhält (Chapter 11).

Teil IV basiert auf Kollaboration mit Gerd Lindner, Michael Hamann, Henning Meyerhenke und Dorothea Wagner. Resultate erschienen als *Structure-Preserving Sparsification of Social Networks* im Tagungsband der *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2015)*. Eine erweiterte Version wurde vor kurzem zur Veröffentlichung in der Zeitschrift *Social Network Analysis and Mining* angenommen

Teil V: Generative Modelle für realistische synthetische Netzwerke

In der experimentellen Algorithmik sind synthetische Instanzen ein Standardwerkzeug. Es sollte jedoch Belege dafür geben, dass auf synthetischen Instanzen beobachtete Resultate repräsentativ sind für das Verhalten auf realen Daten. Das Verhalten von Algorithmen, die auf komplexen Netzwerken operieren, hängt oft stark von der Struktur der Netzwerke ab, deshalb ist dieses Problem hier sowohl besonders relevant als auch kompliziert. Teil V behandelt daher generative Modelle für realistische synthetische Graphen. Verwandte Arbeiten haben bereits eine Vielzahl an Modellen vorgestellt, von denen einige den Anspruch haben, umfassend realistisch zu sein, d.h. in realen komplexen Netzwerken häufig beobachteten Muster gut abzubilden. Wir nähern uns einer Definition von Realismus von zwei verschiedenen Blickwinkeln, als der Replizierung von Strukturen sowie der Replizierung von Laufzeiten.

Ein Ziel ist es Repliken derselben Größe zu erstellen. Darüber hinaus widmen wir uns insbesondere dem Anwendungsszenario, eine skalierte Replik eines gegebenen Originalnetzwerks zu generieren, was in der Literatur bisher wenig betrachtet wurde. Wir stellen geeignete Verfahren vor, um die betrachteten Modelle so zu parametrisieren, dass sie hochskalierte Versionen des Eingabegraphen generieren. Als vielversprechendsten Ansatz stellen wir den LFR+ Generator vor, eine Modifikation des LFR Modells für Community Detection-Benchmarks. Wir machen uns die tatsächliche Flexibilität der bei LFR eingesetzten algorithmischen Methoden zu nutze, um den Realismus der Modellierung zu verbessern. Unsere schnelle Implementierung in NetworKit generiert Graphen anhand des gewöhnlichen LFR-Modells und des erweiterten LFR+-Modells, und tut dies deutlich schneller als die Referenzimplementierung, auch mithilfe von Parallelverarbeitung. Gegenüber seinem Vorgänger zeichnet sich LFR+ durch mehr Flexibilität, Realismus und Effizienz aus. Insbesondere ist LFR+ im Allgemeinen realistischer als die Modelle

RMAT; BTER und Hyperbolic Unit Disk Graphs, die alle als so realistisch gelten, dass sie stellvertretend für reale Daten verwendet werden können. Wir zeigen, dass unser Entwuf ein skalierbares und effektives Werkzeug ergibt, um ein reales Netzwerkes inklusive wichtiger Eigenschaften auf der Mikro- und Makroebene nachzubilden – und eventuell um Größenordnungen zu skalieren. Dies liefert realistische Testdaten für das Algorithm Engineering auf Netzwerken für den Fall, dass geeignete reale Daten nicht zur Verfügung stehen.

Teil V basiert auf einer aktuellen Kollaboration mit Sasha Gutfraind, Ilya Safro, Henning Meyerhenke und Michael Hamann, und wurde bisher nicht publiziert.

Part I

INTRODUCTION

1 NETWORK SCIENCE

> Nan-in, a Japanese master during the Meiji era, received a university professor who came to inquire about Zen. Nan-in served tea. He poured his visitor's cup full, and then kept on pouring. The professor watched the overflow until he no longer could restrain himself. "It is overfull. No more will go in!"
>
> "Like this cup," Nan-in said, "you are full of your own opinions and speculations. How can I show you Zen unless you first empty your cup?"
>
> – traditional Zen koan

The algorithmic methods and software tools presented in this thesis find their main applications in the emerging field of network science. In this introductory chapter, I delineate the field and briefly discuss disciplinary and historical as well as methodological aspects. Network science is interdisciplinary by nature and intersects with multiple academic fields, including sociology, statistics, complex systems studies, and computer science. Several examples from previous literature illustrate the wide range of possible applications of network science methodology. In the following I introduce central terms and concepts and clarify what defines network data, the common input for all methods discussed in the course of this thesis.

1.1 MOTIVATING EXAMPLES

"Networks are everywhere" is the rallying cry of an emerging discipline. As of 2016, a Google Scholar search for the phrase returns over 500 publications. This slogan is an expression of excitement upon discovering that methods of network science are potentially applicable to a vast variety of relevant phenomena, spanning social (e.g. a friendship graph), technical (e.g. the internet or electric power grids), biological (e.g. protein interaction networks) and informational (e.g. the world wide web) domains. At the same time, the slogan is — on a closer look — a sign of a certain immaturity of the discipline. After all, why don't we regularly hear statisticians exclaiming in excitement that "tabular data is everywhere"?

Perhaps it is best to begin with some examples, illustrating the broad applicability of network methods, before delineating the term "network science" more rigorously. Surveys such as [COJT+11] have cataloged some of the broad spectrum of examples from the literature in which networks have been used to map and understand real world systems. To illustrate this, it is worth looking at a few examples in more detail. In the following, I discuss some successful (or at least creative) applications of network science methodology in the study of research questions in different domains.

Early in the history of network science methods it became clear that modeling social systems as a set of actors with ties between them provides a useful perspective on social phenomena. Hence, many methods for the analysis of complex networks were pioneered on *social networks* [Fre04]. This historical association is still strong so that network methods are still frequently but incorrectly subsumed under the term "social network analysis". Nonetheless, social networks are of course highly relevant and intuitive examples for the

methods described in the following, and network analysis has developed into one of the main techniques of *computational sociology*. While in the early days the collection of social network data (e.g. with the help of questionnaires) was troublesome, the current rise of online social networking services allows for social network analysis on an unprecedented scale. Certainly the largest and most prominent example is the massive amount of social network data collected by Facebook. Ugander et al. [UKBM11] first describe a structural profile of the Facebook social graph - containing at the time 721 million individuals and about 70 billion friendship links - using a catalog of common measures (many of which are described in Chapter 2). Among its key properties, they find an average distance of 4.7 hops between individuals (confirming again the *small world phenomenon* in networks, also known as "six degrees of separation" [dSPK79]) and a strongly skewed distribution in the number of friendship links per user (though not, as often claimed, a power law distribution, cf. Sec. 2.3). The structure of the network also leads to an apparent paradox with important psychological implications, namely that "your friends have more friends than you" (i.e. 83.6% of Facebook users have less friends than the median friend count of their friends).

Differentiating between different kinds of ties according to their position in the network structure (cf. Chapter 10 and in particular Sec. 10.1) is an idea that has received attention already very early in the history of social network analysis (e.g. in the famous study on how "weak ties" between distant acquaintances bridge tightly knit communities and are vital for the spreading of novel information [Gra73]). More recently — in an effort to distinguish strong from weak social ties in online social networking data — researchers with access to the Facebook social graph answered the question whether a particular type of tie, namely romantic partnership, can be identified from the network structure alone [BK14]: Given all the connections among a person's friends, they demonstrate that it is possible to identify the person's romantic partner with high accuracy, and without relying on any data other than the network structure. The algorithm that performs this classification task relies on a new measure of tie strength (i.e. *edge centrality*, see Def. 29). The measure, termed *dispersion*, assigns a high rating to a friendship link between two people when their mutual friends are not well connected to one another. This expresses the notion that the romantic partner is likely to have connections into multiple contexts of a person's social life, termed social foci, which are recognizable in the network as densely connected groups of nodes (i.e. *communities*, cf. Chapter 7). The strong performance of their classification algorithm also draws attention to the issue that online social networking services are able to infer a surprising amount of personal information which users have not chosen to disclose explicitly. In this context, social network analysis has gained a political dimension, since it has come to light that intelligence agencies such as the NSA are collecting massive amounts of social network data on the general population.

Another creative application of network science methodology comes from a study that compares characteristics of real social networks with social networks depicted in mythology, in particular three heroic epics, the ancient Greek *Iliad*, the Old English *Beowulf* and the Irish *Táin Bó Cuailnge* [MCK12]. The researchers are looking for quantitative evidence on the question whether these narratives are entirely fictional or likely to contain a historic core. The social networks they contain are compared to real social networks (e.g. from online social networking and scientific collaborations) on the one hand, and imaginary social networks in other works of fiction, formed by characters appearing

together in *Marvel* comic books, in Tolkien's *Fellowship of the Ring* or Hugo's *Les Misérables*, on the other. This comparison is done according to a set of network properties, such as mean degree and clustering coefficient (Sec. 2.3), mean shortest path length and diameter (Sec. 2.2), absolute size of giant component (Sec. 2.4) and degree assortativity (Def. 21). Thereby, they construct and compare structural profiles of the network, similar to but less extensive than the approach described in Sec. 5.3.2 of this thesis. The core idea is that imaginary social networks cannot hide their artificiality while characteristics of historically real social networks are carried on in the story even when fictional elements are introduced. Real social networks have been found to be typically highly clustered, have a low diameter ("small world phenomenon") and positive degree assortativity, which are here assumed to be essential properties. Accordingly, lack of degree assortativity in the *Táin* network and other structural differences are reported as evidence for being imaginary. Playful as this study may be, it does provide a good example of how network theory opens a new perspective and previously unavailable means of quantitative analysis on a well-studied subject.

Complex networks are also of interest in economics, for example the network created by users of the Bitcoin currency. Bitcoin is a decentralized digital currency that is implemented through a peer-to-peer system of software clients, some of which invest computing power to verify transactions cryptographically and ensure the desired properties of the currency. The widespread adoption of Bitcoin has enabled the analysis of financial links and flows with an unprecedented scope since the system maintains a complete public list of all transactions. In the network model of a 2013 study [KPCV14], a node represents a Bitcoin address and a directed edge represents a payment (with an associated time stamp and amount), yielding a graph that grows to ca. 13 million nodes and 44 million edges. Basic network statistics such as degree distribution and clustering (Sec. 2.3) are computed, and their temporal evolution is tracked. The study finds that the network is governed by "rich get richer" dynamics in the literal sense, resulting in a highly heterogeneous wealth distribution. After an initial phase of fluctuations, network properties stabilized to their typical values as Bitcoin "went mainstream" and gained commercial value.

Network models are by no means restricted to social phenomena, in the sense of a set of actors as nodes and their interactions as edges. Among the complex systems for which a network model can be naturally applied is the brain. A *connectome* is a network that maps connections in the brain. The study of the connectome is based on the assumption that understanding the connection patterns of this biological computer is fundamental for neuroscience. Connectome data sets usually fall into the categories of anatomical networks, in which edges represent physical connections between elements of the brain, or functional networks, in which edges are derived from dynamic activation patterns of brain regions (e.g. correlations of activation-level time series) [BS09]. Connectome networks can be defined on different scales, from individual neurons and axons as nodes and edges, to more macroscopic structures such as fiber tracts. Mapping the connectome with high resolution continues to present unresolved challenges for imaging technology. As of 2015, the roundworm *C. elegans* remains the only organism for which the full connectome on the neuronal scale has been reconstructed. With advances in imaging, we can expect to obtain massive complex networks that also reach the boundaries of computing capabilities. It is estimated that the neuronal-scale connectome of the human brain is a graph on the order of 10^{10} nodes and 10^{14} edges [ACG+09]. At various scales, the anatomical network of the

human brain is characterized, for instance, by a low diameter (Def. 8), a skewed degree distribution (cf. Sec. 2.3) and the presence of *connector hubs*, high-degree nodes connected to different communities (cf. Chapter 7). Brain network properties are also being explored as diagnostic markers, e.g. by linking a loss of clustering (Def. 16) in functional brain networks to Alzheimer's disease, and schizophrenia to a loss of hierarchical organization, as reflected by a change in correlation between the two centrality measures (cf. Sec.2.3) node degree and clustering coefficient [BS09]. Connectome analysis can also establish relationships between network-structural measures and cognitive function. For instance, regions found to have high betweenness centrality (Def. 17) in anatomical connectomes are the *precuneus* (linked to episodic memory, visuospatial processing, reflections upon self, and aspects of consciousness), the *insula* (linked to emotion and consciousness), the *superior parietal* (linked to spatial orientation) and the *superior frontal cortex* (linked to self-awareness) [IMSCR+08]. Assuming that communication between brain regions is efficient and follows shortest paths, those regions are in the position to mediate much of the communication. Network science methodology has even been used to study the effect of psychedelic drugs on the brain: After a dose of *psilocybin*, the active ingredient of "magic mushrooms", subjects showed a massive rise in the connectivity and integration of the functional connectome, with many functional connections only present in the psychedelic state [PET+14].

1.2 FOUNDATIONS OF NETWORK SCIENCE

These and many more studies paint a picture of the wide applicability of network analysis methods, but what exactly is at the core of network science? From the preceding examples, one might conclude that it is defined by the mapping of a *complex system* (a brain, an economy, a society...) as a network. The term *complex system* is notoriously difficult to define, but study of complex systems is concerned with understanding systems composed of many parts with nontrivially patterned relationships among them, which give rise to emergent properties of the system as a whole. Science has followed mainly an analytical approach over the course of its history, taking complex phenomena apart by identifying and describing their simpler parts [BY02]. Study of complex systems motivates a shift of perspective towards the interconnections between these parts. Interconnections become "first class citizens" in theories about the system in question. Accordingly, a network approach is especially suited to map and analyze such systems, and we often subsume under "network science" any research that uses graphs to describe complex systems. Still, it is helpful to adopt another perspective that is both more general and more rigorous. According to this perspective, network science is a form of statistics that focuses on relationships between parts as a special form of data. Following the framework defined by Brandes et al. [BRMW13] [Bra16], it is important to clarify what defines *network data* as opposed to other types of data. Network analysis – just as any form of statistics or data analysis – starts with the assignment of values to variables. A variable $x : \mathcal{D} \to \mathcal{R}$ assigns to each element from a domain $\mathcal{D}$ a value in the range $\mathcal{R}$. Possible ranges for the values may differ, as well as the operations defined on them, commonly known as different *scales of measurement* (including nominal, ordinal, interval, and ratio levels). A network data set contains both atomic units of observation (the elements of the phenomenon or system, yielding the nodes of the network) and dyadic units of observation (links, relationships or associations between elements, yielding the edges of the network). What

makes it network data is that the data set contains variables defined on a domain of dyads:

Definition 1 (Network)**.** Let $\mathcal{N}$ be a set of atomic elements. Then a subset $\mathcal{D} \subseteq \mathcal{N} \times \mathcal{N}$ of dyads is called a *dyadic domain*. A *network* is a mapping $x : \mathcal{D} \to \mathcal{R}$ from elements (u, v) of the dyadic domain $\mathcal{D}$ to values $x(u, v)$ from a value range $\mathcal{R} \subseteq \mathbb{R}$.

We call x an *undirected network* if $(u, v) \in \mathcal{D} \iff (v, u) \in \mathcal{D}$ and $x(u, v) = x(v, u)$, and a *directed network* otherwise. For undirected networks, we let $\{u, v\}$ represent both (u, v) and (v, u) for simplicity. □

We consider only *one-mode networks* in the course of this work, but the definition can be extended to cover *two-mode networks*, in which dyads $\mathcal{D} \subseteq \mathcal{N} \times \mathcal{A}$ contain one element each from two disjoint sets (see [BRMW13] [Bra16]). Dyads may be directed or undirected, and in the following we assume dyads to be undirected unless noted. The values a dyadic variable can assume may be binary ($R = \{0, 1\}$, indicating presence or absence of an association) or otherwise valued. Furthermore, it is essential that these dyads intersect via common elements, so that it is meaningful to speak of one network as an entity to analyze.

In order to make a phenomenon accessible to studying it with network science methods (including those described in this thesis), it is essential to form a network model of the phenomenon. This entails two steps: The abstraction of the phenomenon into a network concept, and the representation of aspects of the phenomenon in the form of network data.

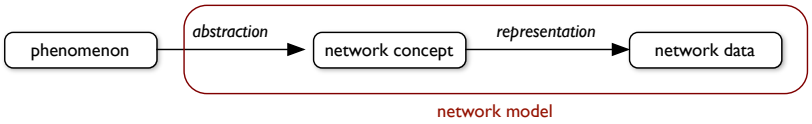


Figure 1: The process of network model formation [BRMW13]

For clarity, let us retrace the process of network model formation in the example of a study on dolphin social networks [LSB+03]: Hoping to gain insights into the complex social behavior of a population of dolphins living in a fiord off the coast of New Zealand, researchers postulated that it can be abstracted to a set of long-lasting companionship ties (not unlike human friendship ties). Here, individual dolphins are treated as the set of atomic units of observation $\mathcal{N}$, and pair-wise associations between dolphins are the dyads $\mathcal{D} \subseteq \mathcal{N} \times \mathcal{N}$ on which the study focuses. This yields the network concept, which motivated the collection of network data through empirical observation. Over a period of 7 years, an association between two dolphins was observed if both were photographed swimming together in a coordinated group (or school). Since an edge of the network should represent long-lasting rather than incidental associations, the observation data was further processed. A weight in the form of $HWI(u, v) = \frac{X}{X+0.5(Y_u+Y_v)}$ was assigned to a dyad $\{u, v\}$, where X is the number of schools where dolphin u and dolphin v were seen together, Y_u is the number of schools where dolphin u was sighted but not dolphin v, and Y_v is the number of schools where dolphin v was sighted but not dolphin u. To

arrive at binary, unweighted network data, in which an association is either present or not present, these weights were tested for significance with respect to a suitable null model in which individuals were randomly permuted within the observed groups. If a dyad $\{u, v\}$ had a weight significantly higher than expected, an edge $\{u, v\}$ in the social network was created.

A network can be naturally represented as a *graph*, a mathematical object consisting of a set of *nodes* (representing the atomic units) and *edges* (representing dyadic variables), so much so that the terms are often used interchangeably. Nonetheless, a distinction between the terms is conceptually useful, referring to a network as the set of observed values associated to overlapping dyads, and a graph as a specific combinatorial representation of this data, among other possible representations (e.g. the adjacency matrix). In this sense, not every possible graph represents a network, but every network has a *graph representation*, defined as follows:

Definition 2 (Graph of a Network)**.** A *graph* $G = (V, E)$ consists of a set of *nodes* V (sometimes called *vertices*), a set of *edges* E (sometimes called *links*), and a *weight function* $\omega : E \to \mathcal{R}$. Given a network $x : \mathcal{D} \subseteq \mathcal{N} \times \mathcal{N}$, its graph $G = (V, E, \omega)$ is defined as

$$\begin{aligned} V &= \mathcal{N} \\ E &= \{(u, v) \in \mathcal{D} : x(u, v) \notin \{0, \infty\}\} \\ \omega(u, v) &= x(u, v) \text{ if } (u, v) \in E \end{aligned}$$

For values from a binary range $R = \{0, 1\}$, the weight function ω is usually omitted. If x is an undirected network (according to Def. 1), we represent it as an *undirected graph* and denote edges as sets $\{u, v\}$ accordingly. In the following definitions, we assume a graph to be undirected unless specified. □

Accordingly, network theory draws heavily from mathematical graph theory, including definitions of concepts, analytical arguments and specifically the formulation of its algorithmic tools, as we shall see in the remainder of the thesis.

Continuing with our example, Fig. 2 shows a node-edge diagram of the graph representing the dolphin social network, including additional attributes. As a small and intuitive example, this network will be used frequently in the course of this thesis to illustrate concepts.

Apart from graphs, network data can also be represented in matrix form:

Definition 3 (Matrix of a Network)**.** Given a network $x : \mathcal{D} \subseteq \mathcal{N} \times \mathcal{N}$, it can be represented as a $|\mathcal{N}| \times |\mathcal{N}|$ matrix M with entries

$$m_{uv} = \begin{cases} x(u, v) & \text{if } (u, v) \in \mathcal{D} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

If values $x(u, v)$ are from a binary range $R = \{0, 1\}$, M is called the *adjacency matrix* of the network. □

The matrix representation allows for the formulation of concepts and algorithms in the language of linear algebra. All of the following definitions could interchangeably be

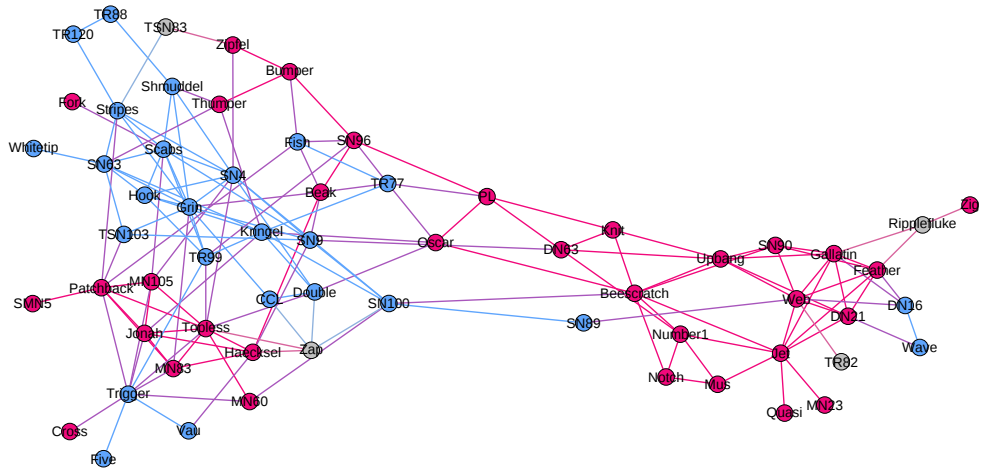


Figure 2: A node-edge diagram of the dolphin social network. Node labels are the names researchers assigned to the individual dolphins. Node color indicates male (pink), female (blue) or unknown (gray).

formulated with either of the mathematical representations. The graph representation is used in most cases, but matrix representation is used where it is arguably more intuitive.

This chapter described network science as being characterized by a distinct kind of data analysis methodology. Clearly, domain-specific knowledge is required to apply network science methodology to a specific research problem, from the formulation of the network model to the collection of network data, from the selection of analytics to the qualitative interpretation of their quantitative results. With this in mind, the perspective that pure network scientists will discover "universal laws" of networks of all possible domains (which they need not be experts on) is perhaps less realistic. Instead – as the field matures, and it becomes common to see the world in network terms – network methods will increasingly find their place – alongside other quantitative methods – in the data analysis tool boxes of researchers from all kinds of areas. But since network science formulates abstractions that are general across problem domains, computer science can contribute general computational tools. This is the context in which the following contributions should be placed.

2 CHARACTERIZING THE STRUCTURE OF NETWORKS

> ACHILLES: Oh, yes, it comes back to me now: the famous Zen koan about Zen Master Zeno. As you say it is very simple indeed.
>
> TORTOISE: Zen Koan? Zen Master? What do you mean?
>
> ACHILLES: It goes like this: Two monks were arguing about a flag. One said, "The flag is moving." The other said, "The wind is moving." The sixth patriarch, Zeno, happened to be passing by. He told them, "Not the wind, not the flag, mind is moving."
>
> – from *Gödel, Escher, Bach*, an endless geek bible

As described in Chapter 1, a considerable part of network science methods focuses on characterizing the structure of a network. This chapter provides a short introduction and classification of quantitative network analysis methods, an extract from the vast catalog of available methods that have been extensively surveyed by, for instance, Newman [New10]. It describes common measures of network structure that are frequently employed throughout the remainder of the thesis.

2.1 GRAPH TERMINOLOGY

Beforehand, recall the definition of a graph representation of a network (Def. 2). We further declare the following useful terminology on graphs:

Definition 4 (Graph (cont.))**.** Let $G = (V, E)$ be a graph. Conventionally we refer to the number of nodes $|V|$ as n and the number of edges $|E|$ as m. The *density* of an undirected graph is $\frac{m}{n \cdot (n-1) \cdot 0.5}$.

A graph $G' = (V' \subseteq V, E' \subseteq E \setminus \{(u, v) : u \notin V' \vee v \notin V'\})$ is called a *subgraph* of G.

Given an edge containing nodes u and v, we say that the edge is *incident* to u and v and both nodes are *adjacent*. In an undirectd graph, the set of nodes adjacent to u is called its (one-hop) *neighborhood* $N(u) = \{v \in V : \{u, v\} \in E\}$. □

Since the algorithmic contributions introduced in Parts III, IV and V apply only to undirected networks, we assume undirected graphs as default in the following definitions. Several of the following concepts can be generalized to directed networks with or without modifying the definition, and the NetworKit tool suite described in Part II is designed to handle directed graphs appropriately by applying the respective variants of common analysis tasks, but discussion of this is omitted here for the sake of brevity.

2.2 DISTANCES IN NETWORKS

Many fundamental measures of a network's structure depend on some notion of *distance* between nodes in a graph. Any *node distance measure* satisfies the following definition:

Definition 5 (Node Distance Measure)**.** Let $G = (V, E)$ be an undirected graph. A node distance measure is a function $d : V \times V \to \mathbb{R}$ with the properties of a metric, i.e.

- non-negativity: $d(u, v) \geq 0$
- coincidence axiom: $d(u, v) = 0 \iff u = v$
- symmetry: $d(u, v) = d(v, u)$
- subadditivity: $d(u, w) \leq d(u, v) + d(v, w)$

□

A natural way to define distance between nodes is in terms of the paths connecting them, and the length of these paths. Of obvious interest is the most cost-effective way of reaching the target node from the source.

Definition 6 (Paths)**.** A path p_{st} is a sequence of edges $(\{u_i, v_i\})$ for $i \in (0, \ldots, k)$ so that $u_0 = s$, $v_k = t$ and $v_i = u_{i+1}$. We call s the *source* and t the *target* node of the path. A path with $s = t$ is called a *cycle*.

Given a non-negative weight function $\omega : E \to \mathbb{R}^+$ that assigns costs to edges, a shortest path σ_{st} is a path with minimum weight $\sum_{e \in p_{st}} \omega(e)$ among all paths p_{st}. In unweighted graphs, this is a path with a minimum number of edges. □

Accordingly, shortest paths yields a straightforward node distance metric:

Definition 7 (Shortest Path Distance)**.** The shortest path distance $d_s(u, v)$ is defined by the length of a shortest path σ_{uv}. □

A value from the distribution of shortest-path distances which is frequently discussed as an important structural property of a network (e.g. under the keyword *small world phenomenon*) is its maximum:

Definition 8 (Diameter)**.** The diameter of a graph G is the maximum shortest-path distance $d_s(u, v)$ among all pairs of nodes in G:

$$\delta(G) := \max_{u,v \in V} d_s(u, v) \tag{2}$$

□

Instead of the extremal value of shortest-path distances, the *effective diameter* is often reported, referring to the distance $\delta_p(G)$ so that the p-quantile of all node pairs has at most this distance. A conventional value for p is 0.9. The distribution of shortest path distances in unweighted graphs has been studied under the term *neighborhood function* [BRV11].

Definition 9 (Neighborhood Function)**.** The neighborhood function ν_G of a graph $G = (V, E)$ yields for each $k \in (1, \ldots, \delta(G))$ the number of pairs of nodes that can be connected in less than k hops.

$$\nu_G(k) : \begin{cases} \mathbb{N} \to \mathbb{N} \\ k \mapsto |\{\{u, v\} \in \binom{V}{2} : d_s(u, v) < k|\} \end{cases} \tag{3}$$

□

However, other measures of node distance can be defined, taking into account not only the shortest paths. As a simple example, the overlap of neighborhoods $N(u) \cap N(v)$ is frequently used to quantify distance (or similarity, cf. Sec. 9.4.2) – which is, speaking of unweighted graphs, equivalent to considering the number of paths of length two between u and v. The scope can be widened by considering paths of greater or arbitrary lengths. Such measures become relevant in the context of routing and diffusion processes in which things traveling through the network do not necessarily take just the most cost-effective route. Among these processes is the flow of electricity in a network of electrical poles (= nodes) connected by conductors (= edges), forming circuits (= paths). One such distance measure is known as *effective resistance*, as it has been first defined within this context.

Definition 10 (Effective Resistance)**.** Let $G = (V, E)$ represent an electrical network, and let I_{uv} be the electrical current resulting from a voltage difference ΔV_{uv} applied to u and v. The effective resistance $\rho(u, v)$ is the resistance of a hypothetical single conductor $\{u, v\}$ directly connecting u and v and replacing all other paths p_{uv}, so that I_{uv} flows when ΔV_{uv} is applied. □

Effective resistance is derived from the well known principle that circuits in series are additive in resistance, while circuits in parallel are additive in conductance. Hence nodes connected by many (preferably short or lightweight) paths are close in terms of effective resistance. Understanding this, we can generalize from electrical networks to all networks in which analogous flow processes happen, and even all networks where - from a purely static point of view - this notion of connectedness is meaningful. The same reasoning applies to random-walk based measures of distance, where the existence of many short paths between two nodes leads to a low expected time for a random walk to travel between them:

Definition 11 (Random Walk, Access Time and Commute Time)**.** For an undirected graph $G = (V, E)$, a random walk of length k steps is a stochastic process with random variables $u_0, u_1, \ldots, u_k$ where u_{i+1} is a node chosen uniformly at random from the neighborhood of u_i. The *access time* $\xi(u, v)$ is the expected number of steps a random walk from u takes to reach v. The *commute time* $\kappa(u, v) = \xi(u, v) + \xi(v, u)$ is the expected time for the random walk to reach v and return to u. □

With this understanding, it is not surprising that effective resistance and commute time are equivalent measures of node distance, formally expressed by the equality $2m\rho(u, v) = \kappa(u, v)$ [Ham11].

Algebraic distance [CS11] α is another similar approach to quantifying the structural distance of two nodes u and v in graphs. Its essential property, analogous to effective resistance and commute time, is that $\alpha(u, v)$ decreases with the number of paths connecting u and v as well as with decreasing lengths of those paths. It follows that nodes within the same dense subgraph of the network are close in terms of α. Algebraic distance can also be described in terms of random walks on graphs and, roughly speaking, $\alpha(u, v)$ is low if a random walk starting at u has a high probability of reaching v after few steps. The iterative definition and calculation of α is described in Sec. 5.1.1. We revisit algebraic distance in Chapters 9 and 11 in order to quantify the "range" of edges.

2.3 CENTRALITY

Network analysis methods which provide a ranking of nodes by their structural importance can be categorized as *node centrality measures*. A large variety of indicators have been proposed under this term. (The centiserver.org project [JSYA+15] lists over 100 different examples from the literature.) Their common denominator can be formulated as follows:

Definition 12 (Node Centrality Measure)**.** A node centrality measure is a function $c : V \to \mathcal{A}$ which assigns to each node u an attribute value $x \in \mathcal{A}$ of (at least) ordinal scale of measurement. The assigned value depends on the position of the node u within the network G as defined by a set of edges $E' \subseteq E$. □

The attribute value range $\mathcal{A}$ is typically $\mathbb{N}$ or $\mathbb{R}_{\geq 0}$. We presuppose in the following definitions that the function receives G as an argument and omit it for brevity.

One of the simplest measures that fits this definition is known in graph theory as the degree of a node.

Definition 13 (Node Degree)**.** The degree of a node is a centrality measure which assigns to each node the number of edges incident to it.

$$\deg(u) : \begin{cases} V \to \mathbb{N} \\ u \mapsto |\{u, v\} \in E| \end{cases} \tag{4}$$

□

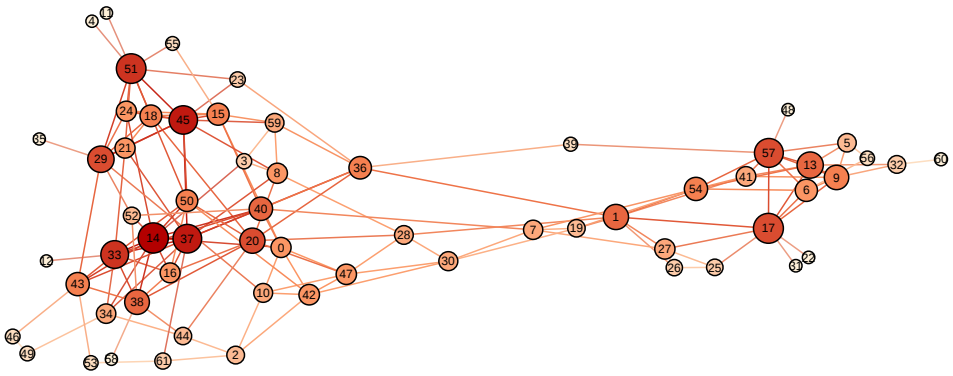


Figure 3: Dolphins social network - node color and size according to node degree

According to *degree centrality*, having more connections makes a node more structurally important. The observation that the distribution of degrees in many real-world complex networks is strongly skewed (e.g. may follow a power law of the form $P(k) \sim k^{-\gamma}$ for degrees k and some characteristic exponent γ) has been highly publicized (e.g. [BA99]), and is certainly a fundamental property of such networks. At the same time, it should be stressed that distributions of other, more complex centralities can be of equal interest - an idea we resume when discussing structural profiles of networks (see Chapter 5).

Like notions of distance, notions of centrality can vary by their scope in the graph. We refer to the scope of the node degree centrality as *local* (or *microscopic*), since only directly incident edges are taken into account. In contrast, the scope of other centrality measures (e.g. betweenness centrality, see Def. 17) is *global* (or *macroscopic*), since every edge of the network potentially influences the centrality score of a node.

The node degree quantifies a very simple idea of centrality in which more local connections mean higher structural importance, without differentiating between different types of edges, or the source of the edge. Accordingly, there is a need for centrality measures that are more discriminating and more global in scope. Among these measures are *eigenvector centrality* and its well-known variant *PageRank*.

Definition 14 (Eigenvector Centrality)**.** Let A be the adjacency matrix of a network and let the nodes $u \in V$ be indexed by $i \in (1, \ldots, n)$. The *eigenvector centrality* of a node u_i is defined as

$$c_e(u_i) : \begin{cases} V \to \mathbb{R}_{\geq 0} \\ u_i \mapsto x_i \end{cases} \tag{5}$$

where the value x_i satisfies the equation

$$x_i = c \sum_{j=1}^{n} A_{ij} x_j \tag{6}$$

□

Rewritten as $\frac{1}{c}x = Ax$, Eq. 6 corresponds to the eigenvector equation $Ax = \lambda x$, hence the vector of centrality values x is an eigenvector of A. Furthermore, the Perron-Frobenius theorem implies that only the greatest eigenvalue $\lambda_{\max}$ satisfies the requirement that all entries in x are positive and thus valid centrality scores. This formalizes the intuitive requirement that the centrality of a node should be proportional to the centrality of its neighbors. An algorithm for a variant of eigenvector centrality has been introduced as PageRank, and is a component of the search engine of Google by providing a ranking of webpages based on hyperlink graph analysis [PBMW99]. A requirement for strict eigenvector centrality is that the graph is connected. This is avoided in PageRank by introducing a probability for random jumps between nodes, which can also be interpreted as adding auxiliary edges to make the graph connected.

Definition 15 (PageRank)**.** Let A be the adjacency matrix of a network and let the nodes $u \in V$ be indexed by $i \in (1, \ldots, n)$. Also, let α be a constant in the interval $(0, 1)$. The *PageRank centrality* of a node u_i is defined as

$$c_p(u_i) : \begin{cases} V \to \mathbb{R}_{\geq 0} \\ u_i \mapsto x_i \end{cases} \tag{7}$$

where the value x_i satisfies the equation

$$x_i = \alpha \sum_{j=1}^{n} \frac{A_{ji}}{\deg(u_j)} x_j + \frac{1-\alpha}{n} \tag{8}$$

□

In Eq. 8, α is called damping factor and represents the probability that the user will continue in the random walk. Equivalently, $1-\alpha$ is the probability of jumping to a random node. Also, notice that if $\deg(u_j) = 0$, we will assume a degree of 1. In the example in Fig. 4, a strong correlation between degree and PageRank is apparent, which is typical, though not trivially true.

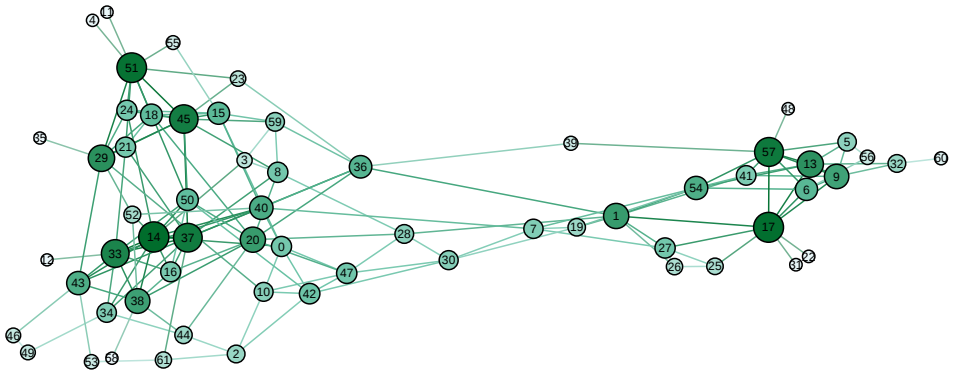


Figure 4: Dolphins social network - node color and size according to PageRank

The *local clustering coefficient* is a centrality index that is focused on connections among a node's neighbors:

Definition 16 (Local Clustering Coefficient)**.** Let $W(v)$ denote the set of edge pairs (or wedges) $\{\{u,v\},\{v,w\}\}$ containing v as the central node, and $T(v)$ the set of triangles $\{\{u,v\},\{v,w\},\{u,w\}\}$. The local clustering coefficient of v is defined as

$$c_l(v) : \begin{cases} V \to [0,1] \\ v \mapsto \frac{|T(v)|}{|W(v)|} = |T(v)| \cdot \binom{\deg(v)}{2}^{-1} \end{cases} \tag{9}$$

□

A node with high $c_l(u)$ can be considered redundant since its removal leaves many direct connections between its neighbors. For the example in Fig. 5, we see how small tightly knit subgroups of low-degree, highly clustered nodes are pointed out by this centrality measure. Although often treated as belonging to a separate category in the literature, the local clustering coefficient is clearly a centrality measure. The question of which values are "good" or "bad" is one of context-specific interpretation, not of formal definition.

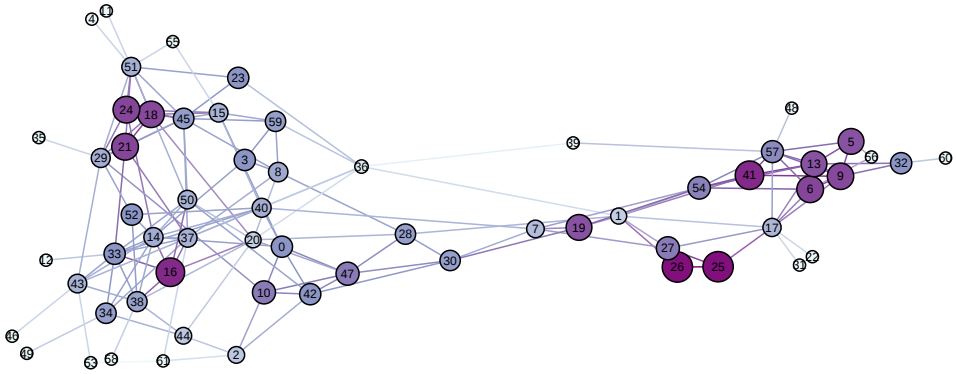


Figure 5: Dolphins social network - node color and size according to local clustering coefficient

Betweenness, another centrality measure that is clearly global in scope, applies especially to networks in which routing or transport processes move something along the network's shortest paths. A node that is positioned on many such paths is considered to be of structural importance, for example because its outage may disrupt the network.

Definition 17 (Betweenness)**.** Naming $|\sigma_{st}|$ the number of shortest paths from a node s to a node t and $|\sigma_{st}(u)|$ the number of shortest paths from s to t that go through u, the (normalized) betweenness centrality of u is defined as [Fre77]:

$$c_b(u) : \begin{cases} V \to \mathbb{R}_{\geq 0} \\ u \mapsto \frac{1}{n(n-1)} \sum_{s \neq u \neq t} \frac{|\sigma_{st}(u)|}{|\sigma_{st}|} \end{cases} \tag{10}$$

□

While it is hard to imagine specific routing processes in a dolphin social network, betweenness can also be interpreted with respect to the static structure of the network: Fig. 6 interestingly shows how betweenness points out one animal in particular whose companionship links place it in an intermediate position between two densely linked subgroups (i.e. communities) of this dolphin population. Note how this individual did not stand out with respect to the other centrality measures.

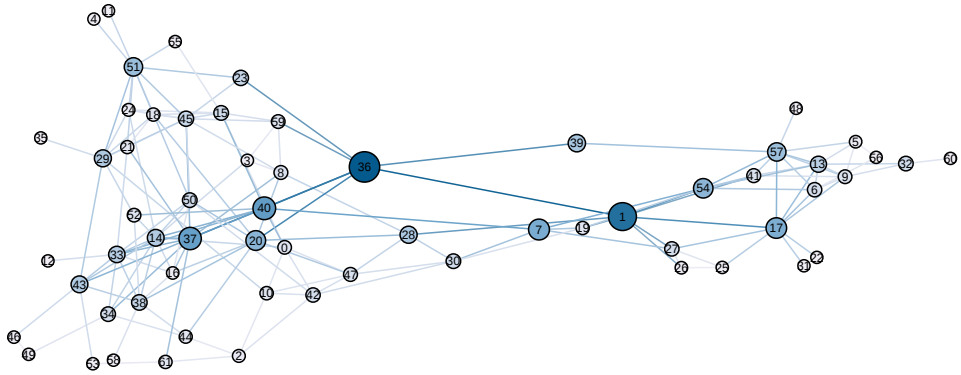


Figure 6: Dolphins social network - node color and size according to betweenness

This section has listed some of the most commonly applied and fundamental centrality measures, but the possibilities of mathematically describing the importance of a node depending on how it is embedded into the network are countless. At the same time, questions about the applicability and interpretation of any centrality measure - whether common, uncommon or defined ad-hoc - must be answered within the context of the network phenomenon that is being studied. Therefore we should not subscribe to the idea that there is a fixed set of measures that are per se relevant in network analyses.

Analogously to centralities on nodes, we can differentiate and rank *edges* by their position within the network's structure, yielding *edge centrality measures*. This concept is examined in Chapter 10, with a focus on structure-preserving sparsification of networks.

2.4 PARTITIONING NETWORKS INTO COHESIVE PARTS

Various analysis methods are concerned with breaking down the node set into subsets depending on their connection patterns within the network. Depending on whether these subsets are overlapping or disjoint, the result can be formalized as a *partition* or a *cover* of the node set:

Definition 18 (Partition and Cover)**.** A partition $\zeta = \{C_1, \dots, C_k\}$ of a graph $G = (V, E)$ is a subdivision of the node set V so that each $v \in V$ is contained in exactly one subset C_i. A cover $\eta = \{C_1, \dots, C_k\}$ of G is a subdivision of V so that each $v \in V$ is contained in at least one subset. □

A graph can be partitioned into its subsets of mutually reachable nodes, its *connected components*:

Definition 19 (Connected Component)**.** In an undirected graph G, a connected component is a maximal subgraph C so that for every pair of nodes $\{s, t\}$ with $t \neq s$ there exists a path $p(s, t)$. Detecting the connected components yields a partition of G. □

Here, the common pattern of cohesion is the existence of a path between each pair in the component. Different criteria of commonality lead to different concepts: For instance, a *community* is loosely defined as a set of nodes whose internal connections are dense and whose external connections are sparse. Chapter 7 discusses the concept in more depth, and Chapter 8 presents efficient algorithms to detect communities.

Another subdivision into increasingly cohesive parts is provided by *k-core decomposition*:

Definition 20 (k-Cores and k-Shells)**.** The *k-core* of an undirected graph G is the largest subgraph of G in which every node has degree of at least k within the subgraph. The set of nodes which belong to the k-core but not the $k+1$-core is called the *k-shell*. The *core number* $c_k(v)$ of a node v is the largest k for which v is still contained in a k-core of G. □

In addition to a partition and a cover of the graph, core decomposition also yields a centrality measure because the core number allows a ranking of nodes according to Def. 12. In the example in Fig. 7, we see that the majority of dolphins have the highest core number $k = 4$. This illustrates how core decomposition can be used to discriminate peripheral from well-connected nodes.

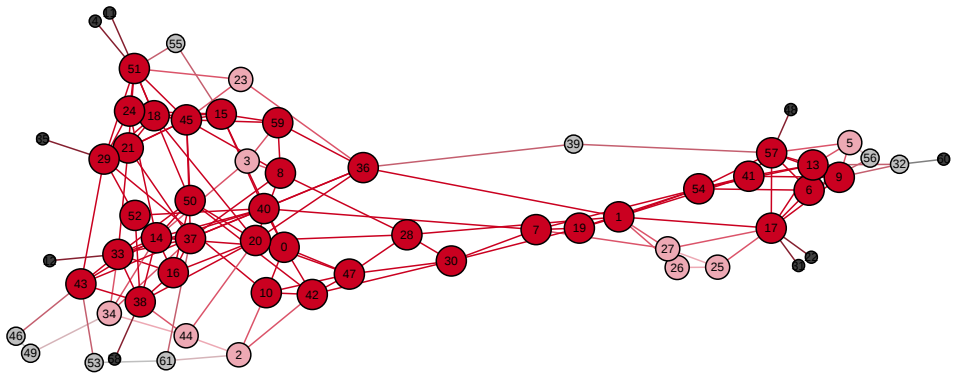


Figure 7: Dolphins social network - node color and size according to core number

2.5 NETWORK-BASED CORRELATIONS

Another frequently applied angle of analysis looks at how structural measures or attributes correlate throughout the network. The most prominent concept of this kind is referred to as *assortativity*: A network is called *assortative* with respect to a node attribute a if there is a positive correlation between a_u and a_v for edges $\{u, v\}$ [New02]. Hence, a network's assortativity coefficient quantifies the tendency of nodes to connect to nodes that are like them with respect to the attribute a. Conversely, the network is called *dissortative* if the correlation is negative, i.e. connections tend to exist between unlike nodes. Synonymously we may speak of *homophily* and *heterophily* in networks.

Definition 21 (Assortativity)**.** For a ratio-scaled node attribute A, let x and y be two vectors of length m where $\{x_u, y_v\}$ are the attribute values for the nodes of edge $\{u, v\}$. Then the assortativity with respect to the attribute is defined as the Pearson correlation coefficient

$$r_{xy} := \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i - \bar{x})^2 \cdot \sum_{i=1}^{m}(y_i - \bar{y})^2}} \tag{11}$$

where $\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$ and $\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i$ are the means of the respective vector.

For a nominal/categorial node attribute B with k different values, let E be a $k \times k$ matrix, with entries e_{ij} representing the fraction of edges connecting nodes of type i to nodes of type j, and the row and column sums a_i and b_i. Then the assortativity coefficient with respect to the nominal attribute B is defined as

$$r = \frac{\sum_i e_{ii} - \sum_i a_i b_i}{1 - \sum_i a_i b_i} \tag{12}$$

□

Network analysis may study the assortativity of external node attributes, or of graph-structural node properties. A simple case of the latter is *degree assortativity*, the correlation coefficient that quantifies the tendency of nodes to connect to nodes with similar degree. However, assortativity with respect to any node centrality measure is potentially a meaningful structural property of the network, as discussed in the context of network

profiles (Chapter 5). For our dolphin social network example, we get a degree assortativity coefficient of -0.043, making the network neither assortative nor dissortative with respect to node degrees.

2.6 EMERGENT PROPERTIES AND SIMULATIONS ON NETWORKS

Beyond characterizing the static structure of a network with measures such as those discussed above, we may also characterize it in terms of the behavior of processes occurring in the network. To the extent that it is determined by the network structure, the behavior of processes can be seen as an emergent property of the network. Processes on networks can be studied through simulation, and epidemic models on networks serve as a prime example: Epidemic models are simplified means of describing the transmission of communicable diseases through a population of individuals, which can intuitively be applied to networks, where edges represent possible infectious contacts between individuals. Studies have recognized the importance of social networks in disease transmission [SKL+10]. One possible model, the SEIR model [KR08], assigns one of four states to each node: Initially one node has been exposed (E) to the infection and all other nodes are susceptible (S). An exposed node becomes infectious (I) after a number of time steps. At each time step, an infectious node contacts all of its neighbors, and with a certain transmission probability, a susceptible neighbor becomes exposed. Nodes stay infectious for a given number of steps, and are then removed (R), either by immunization or death. Counting the number of nodes of each state at each time step yields epidemic curves that describe the dynamics of the outbreak. There is a nontrivial relationship between network structure and epidemic dynamics (e.g., they have been connected to spectral properties of the graph [WCWF03]). Experiments running an SEIR epidemic model are used to provide additional data for the characterization of networks in Chapters 11.

3 OBJECTIVES AND METHODOLOGY

> A novice was trying to fix a broken Lisp machine by turning the power off and on. The master, seeing what the novice was doing, spoke sternly: "You cannot fix a machine by just power-cycling it with no understanding of what is going wrong."
> The master turned the machine off and on. The machine worked.
>
> – traditional hacker koan

This chapter clarifies general objectives of the research presented in this thesis. Subsequently we discuss methodological questions that apply throughout.

3.1 GENERAL OBJECTIVES

Contributions presented in this thesis fall into one or several of the following areas of improvement:

Performance and Scalability: Since promising large network data sets are increasingly common, it is an active current research project to develop scalable methods for the analysis of large networks. Naturally, most analysis methods have been pioneered on small networks (e.g. social networks). Nowadays, various driving factors (among them large-scale web applications, online social networking, ...) contribute to the widespread availability of large network data sets, some examples of which are shown for size comparison in Fig. 8. There is consequently a growing demand for highly scalable analysis algorithms and efficient implementations. In a general sense, we define scalability as the capability to effectively apply algorithms to the growing volume of data that arises in practice. Scalability in the special sense of good parallel scaling behavior (speedup with the number of processors) is a factor in general scalability, and has also been the focus of our engineering work. For each contribution we follow at least one of three paths to achieve higher scalability: algorithmic improvements (finding computational "shortcuts" to a solution), parallelization (distributing computation to multiple processors), as well as heuristics and approximations (yielding an inexact but qualitatively similar result, e.g. through statistical methods). For example, Chapter 8 introduces a parallelization of a heuristic for the modularity optimization problem which enables community detection in larger networks in shorter time, given the resources of a typical multicore workstation. Its implementation in NetworKit is among the fastest currently available.

Effectiveness and Practicality: We put a strong focus on the suitability of algorithms as practical data analysis tools. Achieving computational speedups, i.e. computing a solution to a formally defined problem faster, is therefore rarely the sole focus. Extensive experimental analysis and comparison is devoted to the question of suitability and effectiveness for a practical purpose, e.g. in order to identify problems with previously proposed solutions and to subsequently develop more capable methods. For example, Chapter 11 clarifies the effectiveness of edge sparsification methods, and Chapters 12 and 13 of generative models.

Conceptualization and Systematization: Network science is often introduced as a loose collection of quantitative methods, perhaps an indication of a field in a stage of for-

mation rather than a stage of maturity. It is therefore a current project to formulate conceptual frameworks by identifying connections, commonalities, taxonomies, and abstractions among these methods. For instance, such abstractions are useful in software architecture. They have frequently resulted in proper modularizations which enhance the value of NetworKit as a software framework. An example of this is provided in Chapter 11, where it is shown how a loose collection of sparsification methods from literature can be unified, compared and implemented as local or global filtering by edge centrality scores.

Usability and Availability: Interdisciplinary by nature, network science needs to continually overcome disciplinary barriers. It is a relevant social phenomenon that innovations often stay effectively confined to specialist communities. For instance, a faster network analysis algorithm may have been presented to the algorithmics community through publications in their respective venues, but may not be easily accessible to potential practitioners from various sciences. The NetworKit project aims to provide a short path from algorithm research to ready-to-use software, through development in an open-source model with free software licensing, user-friendly packaging and distribution, and integration into a preexisting ecosystem of software tools.

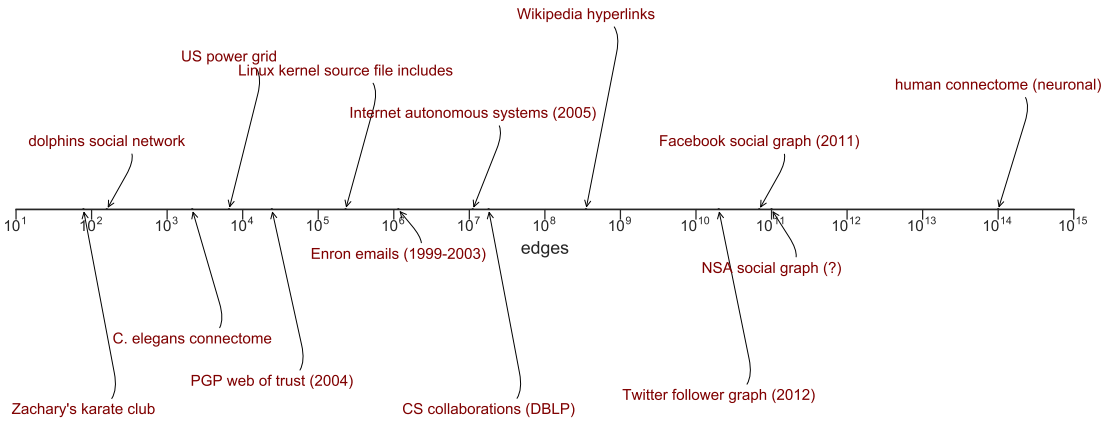


Figure 8: Comparision of network sizes

3.2 ALGORITHM ENGINEERING AND EXPERIMENTAL ALGORITHMICS

Throughout this work *algorithm engineering* [San09] methodology is applied. Algorithm engineering is defined by an iterative approach to algorithm development, cycling through the stages of algorithm design, theoretical analysis, implementation and experimental evaluation. A strong focus is put on the experimental part. For instance, running time analysis does not just take the form of deriving asymptotical, worst-case bounds, but includes reporting of absolute running times on real inputs. The algorithm engineering approach is particularly pertinent to the field of complex network analysis. Firstly, in many instances we consider heuristics that are less amenable to rigourous theoretical analysis, e.g. with respect to computational complexity or result quality guarantees. To the extent that the algorithms described in this thesis are variants or efficient implementations of

known algorithms, theoretical results like running time bounds are treated briefly and referenced in previous work. Secondly, algorithm performance is often highly dependent on the structure of the input. Consequently, it is hard and often ill-advised to generalize from particular networks to complex networks in general. Furthermore, observed performance differences are often hard to trace back definitively to particular structural features. It is therefore crucial to perform experimental evaluation on representative real inputs and realistic models, and do so on an adequately large scale.

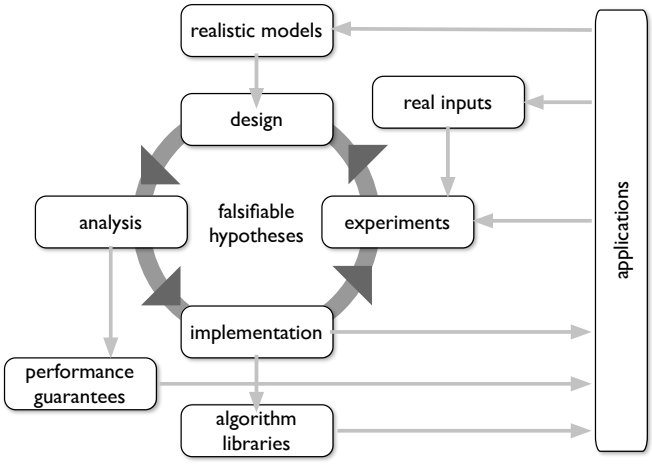


Figure 9: Basic elements of the algorithm engineering methodology [San09]

Sanders, while obviously not arguing against basic research and free theoretical exploration of algorithmic concepts, stresses the need for algorithm research to be grounded in real applications, criticizing "[...] those papers that begin with a vague claim of relevance to some fashionable application area before diving deep into theoretical constructions that look completely irrelevant for the claimed application. Often this is not intentionally misleading but more like a game of 'Chinese whispers' where a research area starts as a sensible abstraction of an application area but then develops a life of itself, mutating into a mathematical game with its own rules." This comment seems especially pertinent in the context of network science algorithmics, where a quick reference to the vast amount of possible areas of real-world application is often considered motivation enough. Algorithmics, where the main object of study is a method, and the possibilities of creating novel methods are limitless, could certainly benefit from stronger feedback to applications. Real-world computational problems should guide researchers in their decisions about which methods to work on, and help to clearly decide which methods are most effective for solving them.

Experimental work plays a crucial role in the algorithm engineering process. According to David Johnson [Joh02], different types of work in experimental algorithmics can be distinguished, including:

1. The *horse race* paper, which aims to prove the superiority of an algorithm, based on standard benchmarks and in comparison with previous competitors.
2. The *experimental analysis* paper, which aims to better understand the strengths and weaknesses of algorithmic ideas in practice.

3. The *experimental average-case* paper, which generates conjectures about the average-case behavior of algorithms where direct probabilistic analysis is too hard.
4. The *experimental mathematics* paper, where the objective is to study the properties of the output of algorithms.

These types apply to different extents to the work presented in this thesis: Chapter 8 is built around a classic horse race, where parallelizations and fast implementations of algorithms for a well-defined optimization problem are evaluated, with previous benchmarks and readily comparable competitors existing. (Though in the context of the "big picture" problem of community detection as a data mining task, the restriction to a single result quality measure enables a horse race but masks certain problems, as briefly discussed in Chapter 7). Still, the horse race paper is perhaps an all too common template, beginning with a novel algorithm idea and ending with the inescapable claim of having significantly improved the state of the art in some respect. This may cause some work - to continue with the equestrian metaphor - to jump the gun and make claims of superiority before proper standards of comparison have been established, and before something that could be considered a "state of the art" (which implies comparability with respect to a single goal) has become recognizable. This is why parts of this thesis that perhaps started out with a horse race in mind have evolved more into the experimental analysis type of work. The literature on selective community detection methods may serve as an example for a collection of methods that may be novel but are poorly understood in comparison to each other, a state of affairs that Chapter 9 tries to correct to some extent. The work on network sparsification in Chapter 11 was conceived as such an experimental analysis from the start, comparing and classifying various existing and novel methods with respect to their behavior, with scalable implementations as a welcome byproduct. While it does present novel algorithmic ideas and performance improvements of existing ones, Chapter 13 is also to a large extent experimental analysis of behavior and practical utility of graph generators, as well as experimental mathematics that seeks to characterize the structure of generated graphs. Throughout these chapters, experimental evidence is used for conclusions about the performance of algorithms, since algorithms on complex networks are not very accessible to a theoretical average-case analysis. However, we should rather speak of typical-case performance, since the notion of an "average" complex network would be nonsensical. We therefore focus on "typical" instances that we consider structurally similar to other real-world instances of high interest, such as social networks, web graphs, internet topology and power grid networks, and connectome data sets.

3.3 REPRODUCIBILITY

The algorithm engineering process raises some interesting questions about experimental design and evaluation. Consider for example a critical argument formulated by Jérôme Kunegis, initiator of the KONECT project [Kun13a]: Suppose we perform experiments to determine wether algorithm X performs better than algorithm Y. Experiments show that algorithm X performs better on 6 out of 10 datasets. Can we conclude the superiority of algorithm X? We must be able to reject the null hypothesis that X and Y win with equal probability on any data set and that results on datasets are independent. In fact, under the null hypothesis, the probability of obtaining the result above is 17 %. Hence the experimental result is not significant. 65 datasets would be needed to get a

statistically significant result (assuming a p-value of ≤ 0.05) for a 60 % outcome. In the case of the graph algorithms we consider, it gets worse: The above argument is based purely on the number of experimental data points and assumes independence of the observations. It does not consider intricate effects that the input graph structure may have on the performance of an algorithm. According to experience, such effects are observed frequently and can be strong, but hard to analyze.

In contrast, it is common in related literature to find empirical results obtained on a more or less careful selection of network data sets to be implicitly or explicitly generalized to "complex networks" or even "graphs" in general. Questions of statistical significance are not usually discussed in experimental algorithmics papers. It may happen that conclusions supported by few experimental data points face criticism in peer review, but this is more often than not based on a rough intuition of plausibility rather than rigorous methodological arguments.

Watchful readers of the remainder of the thesis will observe that this has been written in hindsight and parts are equally guilty of not fulfilling higher standards of statistical reasoning. These are often yet to be determined and universally adopted by the respective research communities. However, some work has already been dedicated to addressing these problems and to formulating best practices (cf. [Joh02], [McG12]). Experimental work presented here is generally following the best practices, as well as avoiding some of the pitfalls pointed out in [Joh02]. So in defense of the empirical conclusions presented in this thesis, let me point out some common principles that were applied when performing experimental algorithmics: To support claims of general applicability to complex networks empirically, experimental data sets are selected to cover a wide range of origins and structural properties (cf. Chapter 8). In some cases it was advisable to restrict the spectrum of networks and conclusions to a specific class of networks with important structural similarities (such as online social networks, cf. Chapter 11). Running time measurements are generally aggregates over a small number of runs, to compensate for possible interference from other running processes, or nondeterminism due to race conditions for parallel computations. When comparing performance of algorithms or implementations, reproducing the behavior of competitors on your own experimental platform is preferable to relying on reported data. Therefore, experimental parts often include re-runs or re-implementations of relevant competitors. The testbeds of instances are compiled from publicly accessible data repositories on the web. Last but not least, implementations to all algorithms presented in this thesis are openly available in source code and as ready-to-use software as part of the NetworKit package, facilitating peer review and reproduction of results.

Part II

NETWORKIT: A TOOL SUITE FOR THE ANALYSIS OF LARGE COMPLEX NETWORKS

4

PRINCIPLES AND ARCHITECTURE

There was a time when rumors began to reach Master Foo and his students of a prodigiously gifted programmer, a young man who wandered the length and breadth of the land performing mighty feats of coding and humiliating all who dared set their skill against his.

Eventually this prodigy came to visit Master Foo, who received him politely and offered him tea. The Prodigy accepted with equal politeness and explained the motive for his visit.

"I have come to you," he said "seeking a code and design review of my latest project. For it is of surpassing complexity, and I do not have peers capable of understanding it. Only an acknowledged master such as yourself" – and here the Prodigy bowed deeply – "can have the discernment required."

Master Foo bowed politely in return and began examining the Prodigy's code. After some time he raised his eyes from the screen. "This code is at first sight very impressive," he said. "It is elegant in design, utilizing original algorithms of great ingenuity, and appears to be implemented in a craftsmanlike way which minimizes the possibility of errors."

The Prodigy looked very pleased at this praise, but Master Foo continued: "However, I detect one significant flaw." "Flaw?" the Prodigy said. "What flaw?" "This code is difficult to read," said Master Foo. "It is only thinly commented, its invariants are not specified, and I see no narrative description of its architecture or internal data structures anywhere. These problems will seriously impede your cooperation with other programmers." The Prodigy drew himself up haughtily. "I do not seek the cooperation of other programmers," he said. "Every time I thought I had found one who might match me in skill I have been disappointed. Thus, I work alone." "But even the hacker who works alone," said Master Foo, "collaborates with others, and must constantly communicate clearly to them, lest his work become confused and lost."“Of what others do you speak?" the Prodigy demanded. Master Foo said: "All your future selves."

Upon hearing this, the Prodigy was enlightened.

– from *The Unix Koans of Master Foo*

Part II introduces NetworKit, an open-source software package for the analysis of large complex networks. NetworKit serves as the software framework on which implementations for all algorithmic contributions presented in this thesis have been built, as well as a suite of tools with which the majority of network analysis results presented have been generated. The design goals of NetworKit focus on high performance on the one hand and flexibility and ease of integration on the other. This is achieved through a hybrid design combining kernels written in C++ with a Python front end for integration into the Python ecosystem of tested tools for data analysis and scientific computing. NetworKit provides the tools to characterize the structure of networks containing up to billions of edges on a typical multicore workstation. A primary goal for the software is to package results of our recent algorithm engineering efforts and put them into the hands of domain experts. The methodology applied to develop scalable solutions to network analysis problems, including techniques like parallelization, heuristics for computationally expensive problems, efficient data structures, and modular software architecture is described in Chapter 4. The package provides a wide range of functionality (including common and novel analytics algorithms and graph generators) for various use cases, discussed in

Chapter 5. In performance benchmarks in comparison with related software (Chapter 6), NetworKit shows the best performance on a range of typical analysis tasks.

A paper describing the package is currently under review for the journal *Network Science*, hence all chapters of this part are based on joint work with coauthors Aleksejs Sazonovs and Henning Meyerhenke. Credit is due to Maximilian Vogel and Michael Hamann for continuous algorithm and software engineering on the package. Furthermore, numerous people have contributed code to the project, namely Lukas Barth, Miriam Beddig, Elisabetta Bergamini, Stefan Bertsch, Pratistha Bhattarai, Andreas Bilke, Simon Bischof, Michele Borassi, Guido Brückner, Mark Erb, Kolja Esders, Patrick Flick, Lukas Hartmann, Daniel Hoske, Gerd Lindner, Moritz v. Looz, Yassine Marrakchi, Mustafa Özdayi, Marcel Radermacher, Matteo Riondato, Klara Reichard, Marvin Ritter, Arie Slobbe, Florian Weber, Michael Wegner and Jörg Weisbarth.

4.1 DESIGN GOALS

Since promising large network data sets are increasingly common in the age of big data, it is an active current research project to develop scalable methods for the analysis of large networks. In order to process massive graphs, we need algorithms whose running time is essentially linear in the number of edges. Many analysis methods have been pioneered on small networks (for the study of social networks prior to the arrival of massive online social networking services), so that underlying algorithms with higher complexity were viable. As we shall see in the following, developing a scalable analysis tool suite often entails replacing them with suitable linear- or nearly-linear-time variants. Furthermore, solutions should employ parallel processing: While sequential performance is stalling, multicore machines become pervasive, and algorithms and software need to follow this development. Within the project, scalable network analysis methods are developed, tested and packaged as ready-to-use software. In this process we frequently apply the following algorithm and software engineering patterns: *parallelization*; *heuristics* or *approximation algorithms* for computationally intensive problems; efficient *data structures*; and *modular* software architecture. With NetworKit, we intend to push the boundaries of what can be done interactively on a shared-memory parallel computer, also by users without in-depth programming skills. The tools we provide make it easy to characterize large networks and are geared towards network science research.

There is a variety of software packages which provide graph algorithms in general and network analysis capabilities in particular (see Section 6.1 for a comparison to related packages). However, NetworKit aims to balance a specific combination of strengths. Our software is designed to stand out with respect to two main aspects:

Performance: Algorithms and data structures are selected and implemented with high performance and parallelism in mind. Some implementations are among the fastest in published research. For example, community detection in a 3.3 billion edge web graph can be performed on a 16-core server with hyperthreading in less than three minutes [SM16].

Usability and Itegration: Networks are as diverse as the series of questions we might ask of them – e.g. what is the largest connected component, what are the most central nodes in it and how do they connect to each other? A practical tool for network analysis should therefore provide modular functions which do not restrict the user to predefined

workflows. An interactive shell, which the Python language provides, is one prerequisite for that. While NetworKit works with the standard Python 3 interpreter, calling the module from the IPython shell and Jupyter Notebook HTML interface [PGO13] allows us to integrate it into a fully fledged computing environment for scientific workflows, from data preparation to creating figures. It is also easy to set up and control a remote compute server. As a Python module, NetworKit enables seamless integration with Python libraries for scientific computing and data analysis, e.g. pandas for data frame processing and analytics, matplotlib for plotting or numpy and scipy for numerical and scientific computing. For certain tasks, we provide interfaces to specialized external tools, e.g. Gephi [BHJ09] for graph visualization.

4.2 ARCHITECTURE

In order to achieve the design goals described above, we implement NetworKit as a two-layer hybrid of performance-aware code written in C++ with an interface and additional functionality written in Python. NetworKit is distributed as a Python package, ready to be used interactively from a Python shell, which is the main usage scenario we envision for domain scientists. The code can be used as a library for application programming as well, either at the Python or C++ level. Throughout the project we use object-oriented and functional concepts. Shared-memory parallelism is realized with OpenMP, providing loop parallelization and synchronization constructs while abstracting away the details of thread creation and handling. The roughly 45,000 lines of C++ code include core implementations and unit tests. As illustrated in Figure 10, connecting these native implementations to the Python world is enabled by the Cython toolchain [BBC+11]. Currently we use Cython to integrate native code by compiling it into a Python extension module. The Python layer comprises about 4,000 lines of code. The resulting Python module `networkit` is organized into several submodules for different areas of functionality, such as community detection or node centrality. A submodule may bundle and expose C++ classes or exist entirely on the Python layer.

4.3 FRAMEWORK FOUNDATIONS

As the central data structure, the `Graph` class implements a directed or undirected, optionally weighted graph using an adjacency array data structure with $O(n+m)$ memory requirement for a graph with n nodes and m edges. Nodes are represented by 64 bit integer indices from a consecutive range, and an edge is identified by a pair of nodes. Optionally, edges can be indexed as well. This approach enables a lean graph data structure, while also allowing arbitrary node and edge attributes to be stored in any container addressable by indices. While some algorithms may benefit from different data layouts, this lean, general-purpose representation has proven suitable for writing performant implementations. In particular, it supports dynamic modifications to the graph in a flexible manner, unlike the *compressed sparse row* format common in high-performance scientific computing. Fig. 11 illustrates the basic layout of the graph data structure schematically, given a small undirected, unweighted graph. Node indices from the range $0 \dots z$ address a collection of arrays (implemented as `std::vector`): ex stores a boolean that marks a node as deleted, so it is skipped during iteration; deg stores the degree of a node for $O(1)$-time access; adj) stores for each node u the indices of its neighbors v, and in this

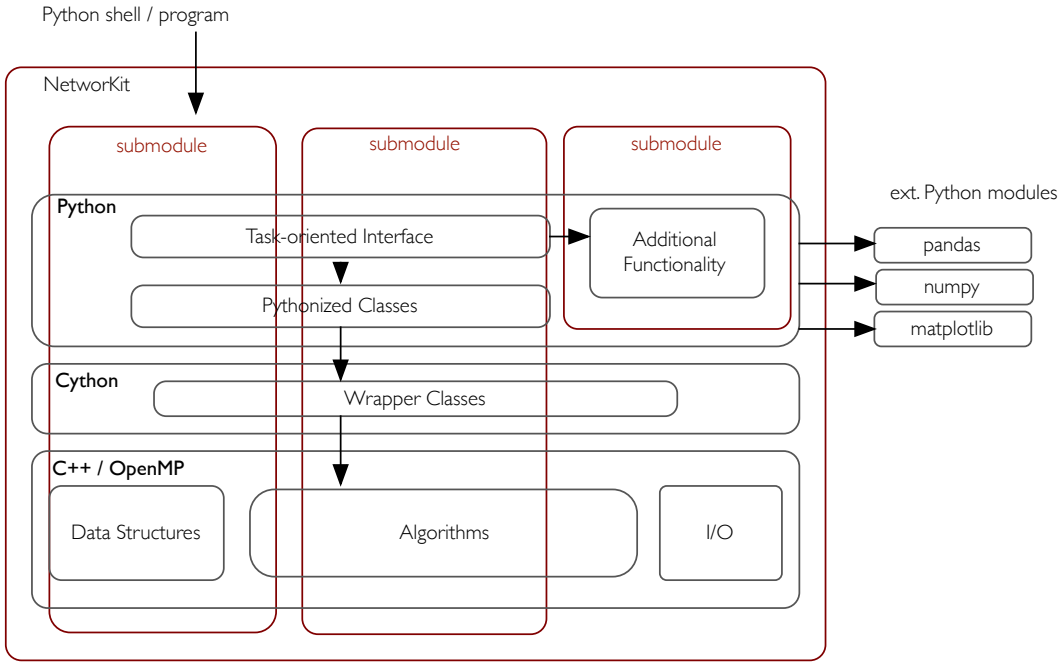


Figure 10: NetworKit architecture overview (→ represents call from/to)

undirected graph a symmetric entry for u must exist in the adjacencies of v; optionally, eid stores for each of the adjacencies (u, v) an edge identifier. Our graph API aims for an intuitive and concise formulation of graph algorithms on both the C++ and Python layer (see Fig. 13 for an example).

4.4 ALGORITHM AND IMPLEMENTATION PATTERNS

Our main focus are scalable algorithms in order to support network analysis on massive networks. We identify several algorithm and implementation patterns that help to achieve this goal and present them below by means of case studies. For experimental results we express processing speed in "edges per second", an intuitive way to aggregate real running time over a set of graphs and normalize by graph size.

4.4.1 *Parallelism*

Our first case study concerns the *core decomposition* of a graph, which allows a fine-grained subdivision of the node set according to connectedness.
More formally, the k-core is the maximal induced subgraph whose nodes have at least degree k. The decomposition also categorizes nodes according to the highest-order core in which they are contained, assigning a *core number* to each node (the largest k for which the node belongs to the k-core). The sequential kernel implemented in NetworKit runs in $O(m)$ time, matching other implementations [BZ11].

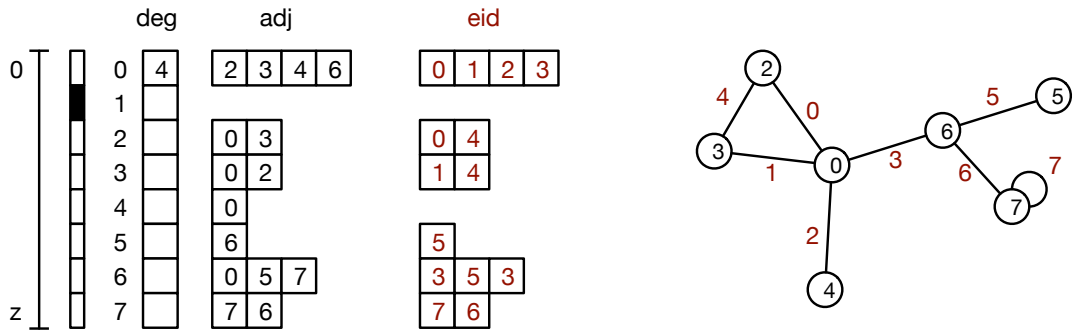


Figure 11: Schematic of `networkit.Graph`: example graph (undirected, unweighted, indexed) and data structure content

The main algorithmic idea we reuse for computing the core numbers is to start with $k = 0$ and increase k iteratively. Within each iteration phase, all nodes with degree k are successively removed (thus, also nodes whose degree was larger at the beginning of the phase can become affected by a removal of a neighbor). Our implementation uses a bucket priority queue. From this data structure we can extract the nodes with a certain minimum residual degree in amortized constant time. The same time holds for updates of the neighbor degrees, resulting in $O(m)$ in total.

While the above implementation already scales to large inputs, it can still make a significant difference if a user needs to wait minutes or seconds for an answer. Thus, we also provide a parallel implementation. The sequential algorithm cannot be made parallel easily due to its sequential access to the bucket priority queue. For achieving a higher degree of parallelism, we follow [DRZ14]. Their ParK algorithm replaces the extract-min operation in the above algorithm by identifying the node set V' with nodes of minimum residual degree while iterating in parallel over all (active) nodes. V' is then further processed similarly to the node retrieved by extract-min in the above algorithm, only in parallel again. ParK thus performs more sequential work, but with thread-local buffers it relies on a minimal amount of synchronization. Moreover, its data access pattern is more cache-friendly, which additionally contributes to better performance.

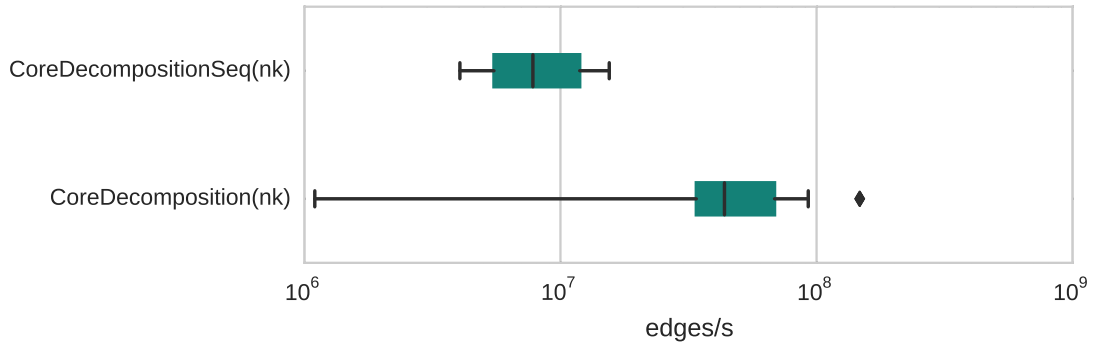


Figure 12: Core decomposition: sequential versus parallel performance

Fig. 12 is the result of running time measurements on a test set of networks (see Sec. 6.2 for the setup). We see that on average, processing speed is increased by almost an order of magnitude through parallelization. Some overhead of the parallel algorithm implies that speedup is only noticeable on large graphs, hence the large variance. For example, processing time for the 260 million edge uk-2002 web graph is reduced from 22 to 2 seconds.

4.4.2 *Heuristics and Approximation Algorithms*

In this example we illustrate how inexact methods deliver appropriate solutions for an otherwise computationally impractical problem. *Betweenness centrality* is a well-known node centrality measure that has an intuitive interpretation in transport networks: Assuming that the transport processes taking place in the network are efficient, they follow shortest paths through the network, and therefore preferably pass through nodes with high betweenness. For instance, their removal would interfere strongly with the function of the network. It is clear that network analysts would like to be able to identify such nodes in networks of any size. NetworKit comes with an implementation of the currently fastest known algorithm for betweenness [Bra01], which has $O(nm)$ running time in unweighted graphs.

With a closer look at the algorithm, opportunities for parallelization are apparent: Several single-source shortest path searches can be run in parallel to compute the intermediate *dependency* values whose sum yields a node's betweenness. Figure 13 shows C++ code for the parallel version, which is simplified to focus on the core algorithm, but the actual implementation is similarly concise. To avoid race conditions, each thread works on its own dependency array, which need to be aggregated into one betweenness array in the end (lines 35-39).

We now evaluate the performance of the implementations experimentally (see Section 6.2 for settings). Figure 14 shows aggregated running speed over a set of smaller networks (from Table 4). In practice, this means that the sequential version of Brandes' algorithm (BetweennessSeq) takes almost 8 hours to process the 600k edge graph caidaRouterLevel (representing internet router-level topology [CAI03]). Parallelism with 32 (hyper)threads (Betweenness) reduces the running time to ca. 90 minutes. Still, parallelization does not change the algorithm's inherent complexity. This means that running times rise so steeply with the size of the input graph that computing an exact solution to betweenness is not viable on the large networks we want to target. In typical use cases, obtaining exact values for betweenness is not necessary, though. An approximate result is likely good enough to appreciate the structure of the network for exploratory analysis, and to identify a set of top betweenness nodes. Therefore, we use a heuristic approach based on computing a relatively small number of randomly chosen shortest-path trees [GSS08]. In contrast to the exact algorithm, running the approximative algorithm with 42 samples takes 6 seconds sequentially. Applying this algorithm cuts running time by orders of magnitude, but still yields a ranking of nodes that is highly similar to a ranking by exact betweenness values. We observe that the distribution of relative rank errors (exact rank divided by approximated rank) has little variance around 1.0. Nodes on average maintain the rank they would have according to exact betweenness even with such a small number of samples. Concretely we see, for instance, that the top ten nodes in the exact betweenness ranking are $(3, 14, 22, 2, 58, 10, 54, 39, 127, 55)$ and $(14, 3, 22, 58, 2, 55, 10, 127, 26, 6)$

```cpp
// thread-local scores for efficient parallelism
count maxThreads = omp_get_max_threads();
count z = G.upperNodeIdBound();
std::vector<std::vector<double>> scorePerThread(maxThreads, std::vector<double>(z));

auto computeDependencies = [&](node s) {
    std::vector<double> dependency(z, 0.0);
    // run SSSP algorithm and count paths
    std::unique_ptr<SSSP> sssp;
    if (G.isWeighted()) {
        sssp.reset(new Dijkstra(G, s, true, true));
    } else {
        sssp.reset(new BFS(G, s, true, true));
    }
    sssp->run();
    // compute dependencies for nodes in order of decreasing distance from s
    std::vector<node> stack = sssp->getStack();
    while (!stack.empty()) {
        node t = stack.back();
        stack.pop_back();
        for (node p : sssp->getPredecessors(t)) {
            double w = sssp->numberOfPaths(p)/sssp->numberOfPaths(t);
            double c = w * (1 + dependency[t]);
            dependency[p] += c;
        }
        if (t != s) {
            scorePerThread[omp_get_thread_num()][t] += dependency[t];
        }
    }
};

// iterate over nodes in parallel and apply
G.balancedParallelForNodes(computeDependencies);

// add up all thread-local values
for (auto& localScore : scorePerThread) {
    G.parallelForNodes([&](node v){
        scoreData[v] += localScore[v];
    });
}
```

Figure 13: Code example: Parallel calculation of betweenness centrality

in the approximate ranking. Experiments of this type (see [GSS08]) confirm that in typical cases betweenness can be closely approximated with a relatively small number s of shortest-path searches. Therefore we can replace an $O(nm)$ algorithm with one of time complexity $O(sm)$ in many use cases. The inexact algorithm offers the same opportunities for parallelization, yielding additional speedups: In the example above, parallel running time is down to 1.5 seconds on 32 (hyper)threads.

If a true approximation with a guaranteed error bound is desired, NetworKit users can apply another inexact algorithm [RK15] which accepts an error bound parameter ϵ. It sacrifices some computational efficiency but allows a proof that the resulting betweenness scores have at most $\pm\epsilon$ difference from the exact scores (with a user-supplied probability).

4.4.3 *Modular Design*

In terms of software design, we aim at a modular architecture with proper encapsulation of algorithms into software components (classes and modules). This requires extensive

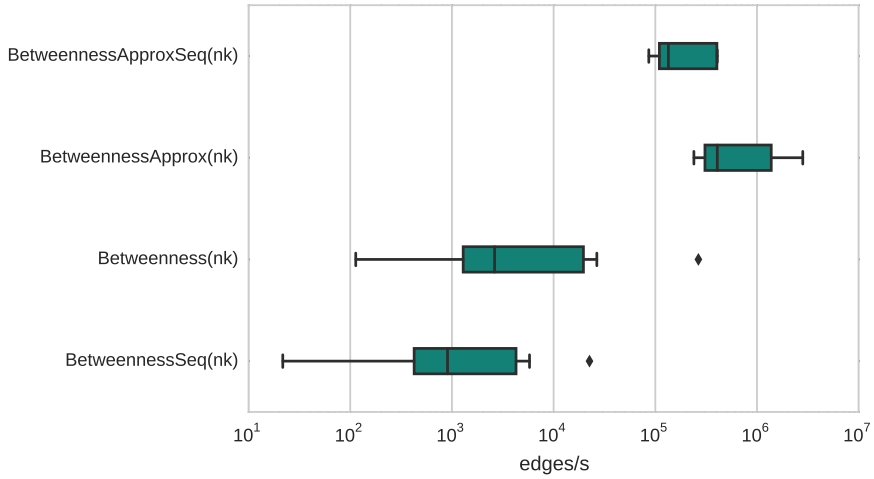


Figure 14: Processing speed of exact and inexact algorithms for betweenness centrality

software engineering work but has clear benefits. Among them are extensibility and code reuse: For example, new centrality measures can be easily added by implementing a subclass with the code specific to the centrality computation, while code applicable to all centrality measures and a common interface remains in the base class. Through these and other modularizations, developers can add a new centrality measure and get derived measures almost "for free". These include for instance the *centralization index* [Fre79] and the *assortativity coefficient* [Fre79], which can be defined with respect to any node centrality measure and may in each case be a key feature of the network.

Modular design also allows for optimizations on one algorithm to benefit other client algorithms. For instance, betweenness and other centrality measures (such as closeness) require the computation of shortest paths, which is done via breadth-first search in unweighted graphs and Dijkstra's algorithm in weighted graphs, decoupled to avoid code redundancy (see lines 10-14 in Fig. 13).

4.4.4 *Efficient Data Structures*

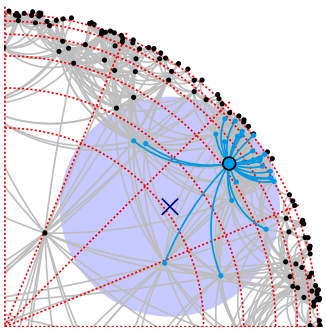


Figure 15: Black: Points corresponding to network nodes. Grey: Edges between nearby points. Red: Boundaries of polar quadtree cells. Blue: Sample node whose neighborhood is visualized by the purple circle and blue edges. (Source: [vLMP15])

The case study on data structures deals with a *generative network model*. Such models are important as they simplify complex network research in several respects (see Section 5.2). Theoretical analyses have shown that *hyperbolic unit-disk graphs* (HUDGs) [KPK+10] have many features also found in real complex networks [BFM14, GPP12, KM15]. The model is based on hyperbolic geometry, into which complex networks can be embedded naturally. During the generation process vertices are distributed randomly on a hyperbolic disk of radius R and edges are inserted for every node pair whose distance is below R. The straightforward HUDG generation process would probe the distance of all pairs, yielding a quadratic time complexity. This impedes the creation of massive networks. NetworKit provides the first generation algorithm for HUDGs with subquadratic running time ($O((n^{3/2} + m) \log n)$ with high probability) [vLMP15]. The acceleration stems primarily from the reduction of distance computations through a polar quadtree adapted to hyperbolic space. Instead of probing each pair of nodes, the generator performs for each node one efficient range query supported by the quadtree. In practice this leads to an acceleration of at least two orders of magnitude. With the quadtree-based approach networks with billions of edges can be generated in parallel in a few minutes [vLMP15]. By exploiting previous results on efficient Erdős–Rényi graph generation [BB05], the quadtree can be extended to use more general neighborhoods [vLM15].

4.5 OPEN-SOURCE DEVELOPMENT AND DISTRIBUTION

Through open-source development we would like to encourage usage and contributions by a diverse community, including data analysts from various fields as well as algorithm engineers. While the core developer team is located at KIT, NetworKit is becoming a community project with a growing number of external users and contributors. The code is free software licensed under the permissive MIT License. The package source, documentation, and additional resources can be obtained from `http://networkit.iti.kit.edu`. The package `networkit` is also installable via the Python package manager `pip`. For custom-built applications, the Python layer may be omitted by building a subset of functionality as a native library.

5 FUNCTIONALITY, IMPLEMENTATIONS AND USE CASES

> A manager asked a programmer how long it would take him to finish the program on which he was working. "It will be finished tomorrow," the programmer promptly replied. "I think you are being unrealistic," said the manager, "Truthfully, how long will it take?" The programmer thought for a moment. "I have some features that I wish to add. This will take at least two weeks," he finally said. "Even that is too much to expect," insisted the manager, "I will be satisfied if you simply tell me when the program is complete." The programmer agreed to this.
>
> Several years later, the manager retired. On the way to his retirement luncheon, he discovered the programmer asleep at his terminal. He had been programming all night.
>
> – from *The Tao of Programming*

This chapter contains a discussion of the set of functionality that NetworKit provides. The package is organized into modules, each bundling several components, e.g. network analysis algorithms, random graph generators, or utility code. We discuss in more detail a core set of algorithms. For selected algorithms, we showcase their efficient implementation. We also discuss possible use cases of NetworKit, on the one hand as an algorithm library in a specialized data analysis pipeline, and a tool for interactive exploratory data analysis on the other. NetworKit's capability to quickly generate and visualize comprehensive structure profiles of large networks aids the user in discovering patterns in network data and formulating hypotheses about them. We discuss a profile of a connectome data set as an example.

5.1 NETWORK ANALYTICS

Table 1 provides an overview of the modular structure of NetworKit. Each of the modules of the package is concerned with an area of functionality, and contains several components. A component is a piece of software that performs a certain task, e.g. computes a network analytic measure. As a general software architecture pattern that is followed, a component is a class, which is instantiated with constructor arguments (e.g. the input graph, algorithm parameters, etc). Computation is triggered by calling a `run` method on the object. Afterwards, "getter" methods are used to retrieve results of the computation.

The following describes the core set of network analysis algorithms implemented in NetworKit. Table 2 summarizes the core set of algorithms for typical problems.

5.1.1 *Distances*

NetworKit contains various components related to distances in networks. Shortest path distances can be computed by breadth-first search (BFS) in unweighted networks and Dijkstra's algorithm in weighted networks. Other properties based on shortest paths include the *diameter* (Def. 8). We use the iFUB algorithm [CGH+13] both for the exact computation as well as an estimation of a lower and upper bound on the diameter. iFub has a worst case complexity of $O(nm)$ but has shown excellent typical-case performance

module	description	example components
algebraic	interface to matrix network representations	adjacencyMatrix
centrality	node centrality measures (Sec. 2.3)	DegreeCentrality PageRank CoreDecomposition
clique	clique finding	MaxClique
coarsening	graph coarsening methods	ParallelPartitionCoarsening (Sec. 8.2.2)
coloring	graph coloring	SpectralColoring
community	community detection (Ch. 7, 8)	PLM Modularity
components	connected components	ConnectedComponents (Sec. 2.4) StronglyConnectedComponents
correlation	correlations in networks (Sec. 2.5)	Assortativity
distance	graph distance measures (Sec. 2.2)	Diameter NeighborhoodFunction AlgebraicDistance
dynamic	support for dynamic graphs	
engineering	tools for algorithm engineering experiments	
flow	graph flow algorithms	EdmondsKarp
generators	generative models (Ch. 12, 13)	LFRGenerator
gephi	import/export for Gephi	
graph	graph data structure and fundamental operations (Sec. 4.3)	Graph BFS
graphio	graph file input/output	GraphMLReader
linkprediction	link prediction (Sec. 5.1.3)	PreferentialAttachmentIndex
matching	graph matching	PathGrowingMatcher
nxadapter	graph conversion for NetworkX	
partitioning	graph partitioning	SpectralPartitioner
profiling	network profiling tool (Sec. 5.3.2)	
sampling	tools for sampling from graphs	
scd	selective community detection (Ch. 9)	PageRankNibble
simulation	simulations on networks (Sec. 2.6)	EpidemicSimulationSEIR
sparsification	network sparsification methods and tools, edge centrality measures (Ch. 10,11)	LocalDegreeScore
structures	fundamental data structures	Partition Cover (Sec. 2.4)

Table 1: Overview of NetworKit modules

category	task	algorithm	time	space
centrality	degree	–	$O(n)$	$O(n)$
	betweenness	[Bra01]	$O(nm)$	$O(n+m)$
	ap. betweeenness	[GSS08],[RK15]	$O(sm)$	$O(n+m)$
	closeness	shortest-path search from each node	$O(mn)$	$O(n)$
	ap. closeness	[EW04]	$O(sm)$	$O(n)$
	PageRank	power iteration	$O(m)$ typical (Sec. 5.1.2)	$O(n)$
	eigenvector centrality	power iteration	$O(m)$ typical	$O(n)$
	Katz centrality	[Kat53]	$O(m)$ typical	$O(n)$
	k-path centrality	[ATK+11]	see [ATK+11]	
	local clustering coefficient	parallel iterator	$O(nd^2)$	$O(n)$
	k-core decomposition	[DRZ14]	$O(m)$	
partitions	connected components	BFS	$O(m)$	$O(n)$
	community detection	PLM, PLP [SM16]	$O(m)$	$O(m)$,$O(n)$
global	diameter	iFub [CGH+13]	$O(m)$ typical	$O(n)$

Table 2: Selection of analysis algorithms contained in NetworKit. Complexity expressed in terms of n nodes, m edges, s samples and maximum node degree d

on complex networks, where it often converges on the exact value in linear time. We also distribute an implementation of a linear-time algorithm for approximating the neighborhood function (Def. 9) [PGF02].

As introduced in Sec. 2.2, algebraic distance [CS11] (α) is a method for quantifying the structural distance of two nodes u and v in graphs. α is computed by performing iterative local updates on d-dimensional "coordinates" of a node. Each node is associated with d initially random values x_u. Then, in each iteration, the coordinates are set to some weighted average of the old coordinates and the average of the old coordinates of all neighbors, according to the following iterative rules, where ω is a relaxation factor:

$$\hat{x}_u^{(k)} = \sum_{v \in N(u)} x_v^{(k-1)} / \deg(u) \tag{13}$$

$$x_u^{(k)} = (1-\omega) \cdot x_u^{(k-1)} + \omega \cdot \hat{x}_u^{(k)} \tag{14}$$

This leads to iterative smoothing so that the coordinates of nearby nodes assimilate. These updates of the node coordinates are parallelized in our code. The algebraic distance is then any distance defined in terms of a norm between the two coordinate vectors. In the following we choose the ℓ_2-norm. As described in [CS11], d can be set to a small constant (e.g. 10) and the distances stabilize after tens of iterations of $O(m)$ running time each. Fig. 16 shows the main section of code for the efficient parallel computation of α in NetworKit. Lines 6-22 and lines 10-15 provide examples of NetworKit's graph iterator

pattern: `G.balancedParallelForNodes` executes the given function (a lambda expression) for each node, while applying dynamic scheduling of blocks of nodes to threads, which is important for load balancing in graphs with heterogeneous degree distributions. `G.forNeighborsOf` iterates over all incident edges/neighbors of a given node.

```
std::vector<double> oldLoads(loads.size());

for (index iter = 0; iter < numIters; ++iter) {
    loads.swap(oldLoads); // store previous iteration

    G.balancedParallelForNodes([&](node u) {
        std::vector<double> val(numSystems, 0.0);
        double weightedDeg = 0;
        // step 1
        G.forNeighborsOf(u, [&](node v, edgeweight weight) {
            for (index i = 0; i < numSystems; ++i) {
                val[i] += weight * oldLoads[v*numSystems + i];
            }
            weightedDeg += weight;
        });

        for (index i = 0; i < numSystems; ++i) {
            val[i] /= weightedDeg;
            // step 2
            loads[u*numSystems + i] = (1 - omega) * oldLoads[u*numSystems + i] + omega * val[i];
        }
    });
}
```

Figure 16: Code example: Parallel computation of algebraic distances.

5.1.2 *Node Centrality*

Node centrality measures quantify the structural importance of a node within a network (cf. Sec. 2.3). The simplest measure that falls under this definition, the *degree* of a node, is stored by the graph data structure for constant-time access. *Eigenvector centrality* and its variant *PageRank* [PBMW99] are implemented in NetworKit based on parallel power iteration, whose convergence time depends on a numerical error tolerance parameter and spectral properties of the network, but is among the fast linear-time algorithms for typical inputs. For *betweenness centrality* we provide the solutions discussed in Sec. 4.4.2. Similar techniques are applied for computing *closeness centrality* exactly and approximately [EW04]. Our current research extends the former approach to dynamic graph processing [BM15, BMS15]. In addition to a parallel algorithm for clustering coefficient, NetworKit also implements a sampling approximation algorithm [SW05], whose constant time complexity is independent of graph size. Given NetworKit's modular architecture, further centrality measures can be easily added.

5.1.3 *Edge Centrality, Sparsification and Link Prediction*

As described in Sec. 10.1 the concept of centrality can be extended to edges: Not all edges are equally important for the structure of the network, and scores can be assigned to edges depending on the graph structure such that they can be ranked. (This requires the edge set to be indexed, which the graph data structure supports optionally.) Such a

ranking can then be used to filter edges and thereby reduce the size of data. NetworKit includes a wide set of edge ranking methods, with a focus on sparsification techniques meant to preserve certain properties of the network. For instance, we show that a method that ranks edges leading to high-degree nodes (hubs) closely preserves many properties of social networks, including diameter, degree distribution and centrality measures. Other methods, including a family of *Simmelian backbones*, assign higher importance to edges within dense regions of the graph and hence preserve or emphasize communities. This topic is the focus of Chapters 10 and 11. While currently experimental and focused on one application, namely structure-preserving sparsification, the design is extensible so that general edge centrality indices can be easily implemented.

A somewhat related problem, conceptually and through common methods, is the task of *link prediction*. Link prediction algorithms examine the edge structure of a graph to derive similarity scores for unconnected pairs of nodes. Depending on the score, the existence of a future or missing edge is inferred. NetworKit includes implementations for a wide variety of methods from the literature [Esd15].

5.1.4 *Partitioning the Network*

Another class of analysis methods partitions the set of nodes into subsets depending on the graph structure, such as *connected components* and *communities*. A network's connected components can be computed in linear time using breadth-first search. We approach community detection from the perspective of modularity maximization and engineer parallel heuristics which deliver a good tradeoff between solution quality and running time [SM16]. The PLP algorithm implements community detection by label propagation [RAK07], which extracts communities from a labelling of the node set. The *Louvain method* for community detection [BGLL08] can be classified as a locally greedy, bottom-up multilevel algorithm. We recommend the PLM algorithm with optional refinement step as the default choice for modularity-driven community detection in large networks. For very large networks in the range of billions of edges, PLP delivers a better time to solution, albeit with a qualitatively different solution and worse modularity. Chapter 8 describes these methods in detail.

5.2 NETWORK GENERATORS

Generative network models aim to explain how networks form and evolve specific structural features, and generate synthetic graphs that resemble real networks in important structural aspects. NetworKit provides a versatile collection of graph generators for this purpose, summarized in Table 3. Chapters 12 discusses different generative models in depth.

5.3 EXAMPLE USE CASES

In the following, we present possible workflows and use cases, highlighting the capabilities of NetworKit as an algorithm library and an interactive data analysis tool.

model [and algorithm]	description
Erdős-Rényi [P. 60] [[BB05]]	random edges with uniform probability
planted partition / stochastic blockmodel	dense areas with sparse connections
Barabasi-Albert [AB02]	preferential attachment process resulting in power-law degree distribution
Recursive Matrix (R-MAT) [CZF04]	power-law degree distribution, small-world property, self-similarity
Chung-Lu [ACL00]	replicate a given degree distribution
Havel-Hakimi [Hak62]	replicate a given degree distribution
Hperbolic Unit-Disk Graphs [KPK+10] [[vLMP15]]	large networks, power-law degree distribution and high clustering
LFR [LF09a]	complex networks containing communities

Table 3: Overview of network generators

5.3.1 *As a Library in an Analysis Pipeline*

A master's thesis [Fli14] provides an early example of NetworKit as a component in an application-specific data mining pipeline (Fig. 17). This pipeline performs analysis of *protein-interaction (PPI) networks.* and implements a preprocessing stage in Python, in which networks are compiled from heterogeneous data sets containing interaction data as well as *expression data* about the occurrence of proteins in different cell types. During the network analysis stage, preprocessed networks are retrieved from a database, and NetworKit is called via the Python frontend. The C++ core has been extended to enable more efficient analysis of *tissue-specific* PPI networks, by implementing in-place filtering of the network to the subgraphs of proteins that occur in given cell types. Finally, statistical analysis and visualization is applied to the network analysis data. The system is close to how we envision NetworKit as a high-performance algorithmic component in a real-world data analysis scenario, and we therefore place emphasis on the toolkit being easily scriptable and extensible.

5.3.2 *Exploratory Network Analysis with Network Profiles*

Making the most of NetworKit as a library requires writing some amount of custom code and some expertise in selecting algorithms and their parameters. This is one reason why we also provide an interface that makes exploratory analysis of large networks easy and fast even for non-expert users, and provides an extensive overview. The underlying module assembles many algorithms into one program, automates analysis tasks and produces a graphical report to be displayed in the Jupyter Notebook or exported to an HTML or LaTeXreport document. Such a *network profile* gives a statistical overview over the properties of the network. It consists of the following parts: First global properties such as size and density are reported. The report then focuses on a variety of node centrality measures, showing an overview of their distributions in the network (see Fig. 18). Detailed views for centrality measures (see Fig. 19) follow: Their distributions are plotted in his-

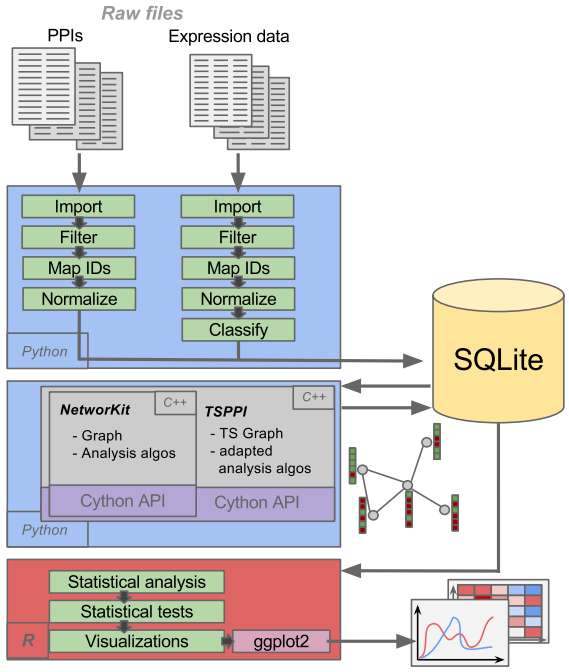


Figure 17: PPI network analysis pipeline with NetworKit as central component

tograms and characterized with standard statistics, and network-specific measures such as centralization and assortativity are shown. We propose that correlations between centralities are per se interesting empirical features of a network. For instance, betweenness may or may not be positively correlated with increasing node degree. The prevalence of low-degree, high-betweenness nodes may influence the resilience of a transport network, as only few links then need to be severed in order to significantly disrupt transport processes following shortest paths. For the purpose of studying such aspects, the report displays a matrix of Spearman's correlation coefficients, showing how node ranks derived from the centrality measures correlate with each other (see Fig. 20b). Furthermore, scatter plots for each combination of centrality measure are shown, suggesting the type of correlation (see Fig. 21a). The report continues with different ways of partitioning the network, showing histograms and pie charts for the size distributions of connected components, modularity-based communities (see Fig. 21b) and k-shells, respectively. Absent on purpose is a node-edge diagram of the graph, since graph drawing (apart from being computationally expensive) is usually not the preferred method to explore large complex networks. Rather, we consider networks first of all to be statistical data sets whose properties should be determined via graph algorithms and the results summarized via statistical graphics. The default configuration of the module is such that even networks with hundreds of millions of edges can be characterized in minutes on a parallel workstation. Furthermore, it can be configured by the user depending on the desired choice of analytics and level of detail, so that custom reports can be generated.

To pick an example from a scientific domain, the human connectome network con-fiber_big maps brain regions and their anatomical connections at a relatively high resolution, yielding a graph with ca. 46 million edges. As the resolution of brain imaging technology improves, connectome analysis is likely to yield ever more massive network

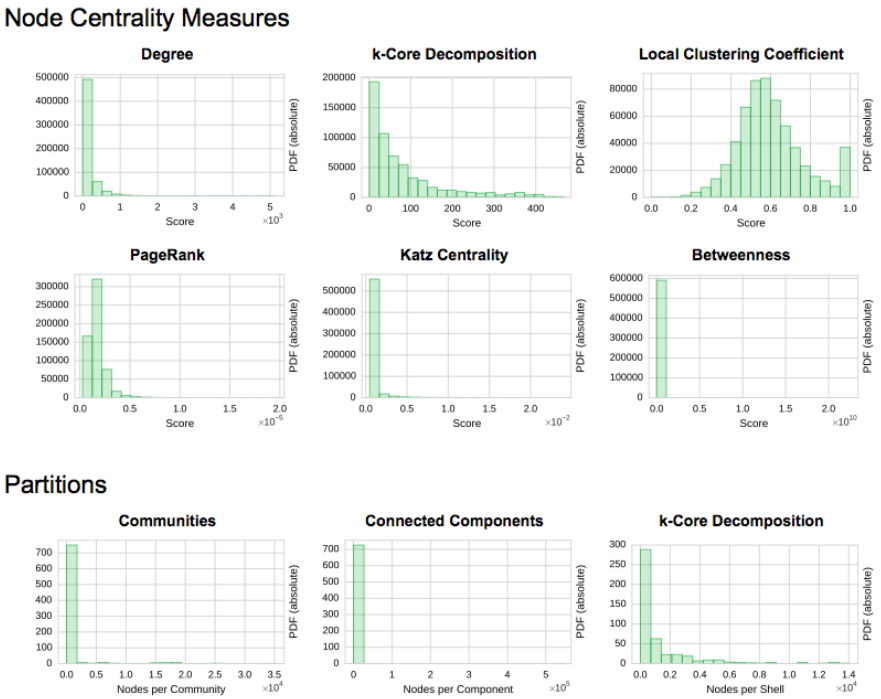


Figure 18: Overview on the distributions of node centrality measures and size distributions of different network partitions – here: a Facebook social network

data sets, considering that the brain at the neuronal scale is a complex network on the order of 10^{10} nodes and 10^{14} edges. On a first look, the network has properties similar to a social network, with a skewed degree distribution and high clustering. The pattern of correlations (Fig. 20b) differs from that of a typical friendship network (Fig. 20a), with weaker positive correlations across the spectrum of centrality measures. As one observation to focus on, we may pick the strong negative correlation between the local clustering coefficient on the one hand and the PageRank and betweenness centrality on the other. High betweenness nodes are located on many shortest paths, and high PageRank results from being connected to neighbors which are themselves highly connected. Thus, the correlations point to the presence of structural hub nodes that connect different brain regions which are not directly linked. Also, a look at a scatter plot generated (Fig. 21a) reveals more details on the correlations: We see that the local clustering coefficient steadily falls with node degree, a majority of nodes having high clustering and low degree, a few nodes having low clustering and high degree. Both observations are consistent with the finding of *connector hub* regions situated along the midline of the brain, which are highly connected and link otherwise separated brain modules organized around smaller *provincial hubs* [SB15].

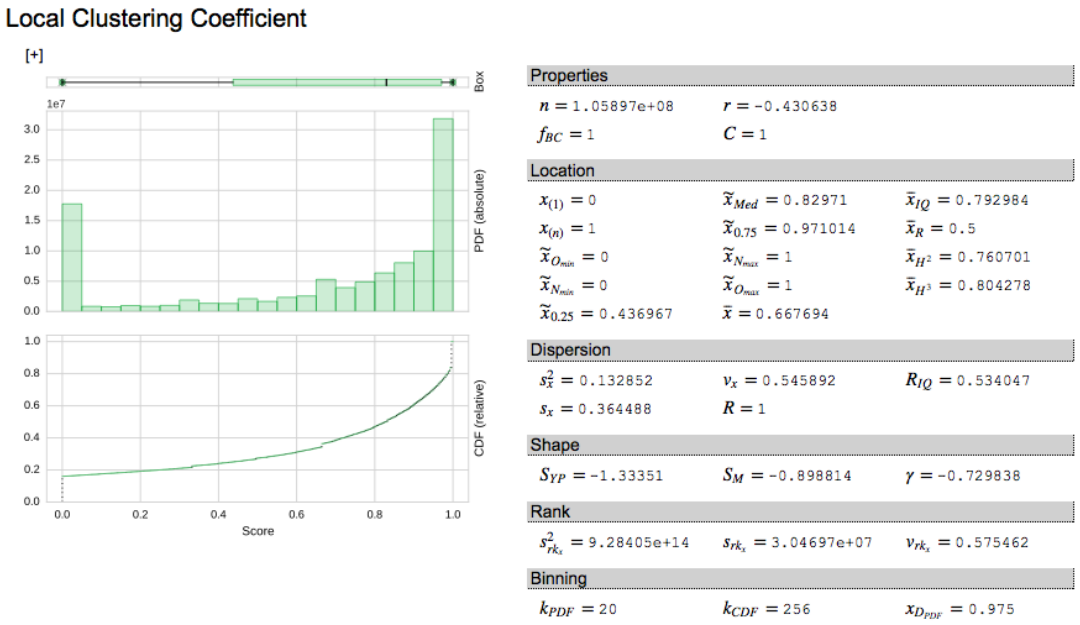


Figure 19: Detailed view on the distribution of node centrality scores – here: local clustering coefficients of the 3 billion edge web graph uk-2007

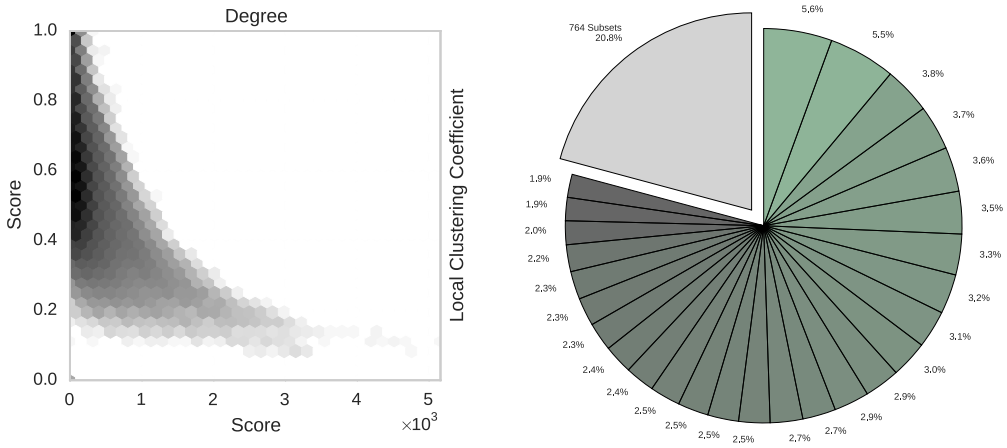
+1.000					Degree
+0.983	+1.000				k-Core Decomposition
−0.376	−0.319	+1.000			Local Clustering Coefficient
+0.993	+0.963	−0.398	+1.000		PageRank
+0.963	+0.966	−0.349	+0.943	+1.000	Katz Centrality
+0.870	+0.826	−0.435	+0.892	+0.852	+1.000 Betweenness

(a) – in the social network fb-Texas84

+1.000					Degree
+0.977	+1.000				k-Core Decomposition
−0.471	−0.322	+1.000			Local Clustering Coefficient
+0.491	+0.329	−0.776	+1.000		PageRank
+0.989	+0.984	−0.411	+0.410	+1.000	Katz Centrality
+0.408	+0.308	−0.659	+0.574	+0.367	+1.000 Betweenness

(b) – in the connectome network con-fiber_big

Figure 20: Correlation between node centrality measures –



(a) Scatter plot of degree and local clustering coefficient

(b) Size distribution of modularity-based communities

Figure 21: Statistical graphics from the profile of the connectome graph con-fiber_big

Another aspect we can focus on is community structure. There has been extensive research on the modular structure of brain networks, indicating that communities in the connectivity network can be interpreted as functional modules of the brain [SB15]. The communities found by the PLM modularity-maximizing heuristic in the con-fiber_big graph can be interpreted accordingly. Their size distribution (Fig. 21b, in which a green pie slice represents the size of a community) shows that a large part of the network consists of about 30 communities of roughly equal size, in addition to a large number of very small communities (grey). Of course, such interpretations of the network profile contain speculation, and a thorough analysis – linking network structure to brain function – would require the knowledge of a neuroscientist. Nonetheless, these examples illustrate how NetworKit's capability to quickly generate an overview of structural properties can be used to generate hypotheses about the network data.

6 COMPARISON AND EVALUATION

> A young monk, new to the temple, not only declared all his methods public but all his instance variables as well. The head abbot grew weary of rebuking the monk and asked the master for advice.
>
> The next day, the master summoned the monk to take lunch with him in his private office. The monk entered to find a sumptuous meal laid out upon the conference table. The master then bade the monk to lie upon the floor. The monk did as commanded, whereupon the master opened the monk's robes and drew a large knife. He pressed the naked point firmly into the monk's chest until a ruby droplet welled up around the blade. The monk cried out in terror and asked the master what his intention was.
>
> "To slit open your belly," explained the master, "so that I may spoon the rice and pour the tea inside. My schedule is quite full, and I find this method of feeding guests to be extremely efficient."
>
> Afterward the monk required no more correction.
>
> – from *The Codeless Code*

This chapter is concerned with evaluating NetworKit in comparison with competing solutions for network analysis. These include existing software packages with similar target use cases, target platforms and design choices. Aspects of the comparison include functionality, user interaction, and supported platforms. For the most relevant competitors, it is shown how NetworKit compares in a performance benchmark on typical network analysis tasks. We also provide a performance comparison with several distributed graph processing frameworks, to show the considerable range of networks that can be analyzed on a single multicore machine before distributed computing solutions are required.

6.1 COMPARISON TO RELATED SOFTWARE

Recent years have seen a proliferation of graph processing and network analysis software which vary widely in terms of target platform, user interface, scalability and feature set. We therefore locate NetworKit relative to these efforts. Although the boundaries are not sharp, we would like to separate network analysis toolkits from general purpose graph frameworks (e.g. Boost Graph Library and JUNG [OFWB03]), which are less focused on data analysis workflows.

As closest in terms of architecture, functionality and target use cases, we see igraph [CN06] and graph-tool [Pei06]. They are packaged as Python modules, provide a broad feature set for network analysis workflows, and have active user communities. NetworkX [HSSC08] is also a mature toolkit and the de-facto standard for the analysis of small to medium networks in a Python environment, but not suitable for massive networks due to its pure Python implementations. (Due to the similar interface, users of NetworkX are likely to move easily to NetworKit for larger networks.) Like NetworKit, igraph and graph-tool address the scalability issue by implementing core data structures and algorithms in C or C++. graph-tool builds on the Boost Graph Library and parallelizes some kernels using OpenMP. These similarities make those packages ideal candidates for an experimental comparison with NetworKit (see Section 6.2.2).

Other projects are geared towards network science but differ in important aspects from NetworKit. Gephi [BHJ09], a GUI application for the Java platform, has a strong focus on visual network exploration. Pajek [BM04], a proprietary GUI application for the Windows operating system, also offers analysis capabilities similar to NetworKit, as well as visualization features. The variant PajekXXL uses less memory and thus focuses on large datasets.

The SNAP [LS14] network analysis package has also recently adopted the hybrid approach of C++ core and Python interface. Related efforts from the algorithm engineering community are KDT [LAB+12] (built on an algebraic, distributed parallel backend), GraphCT [EJRB13] (focused on massive multithreading architectures such as the Cray XMT), STINGER (a dynamic graph data structure with some analysis capabilities and a Python interface) [EMRB12] and Ligra [SB13] (a recent shared-memory parallel library). They offer high performance through native, parallel implementations of certain kernels. However, to characterize a complex network in practice, we need a substantial set of analytics which those frameworks currently do not provide.

Among solutions for large-scale graph analytics, distributed computing frameworks (for instance GraphLab [LBG+12]) are often prominently named. However, graphs arising in many data analysis scenarios are not bigger than the billions of edges that fit into a conventional main memory and can therefore be processed far more efficiently in a shared-memory parallel model [SB13], which we confirm experimentally in a recent study [KSVM15]. Distributed computing solutions become necessary for massive graph applications (as they appear, for example, in social media services), but we argue that shared-memory multicore machines go a long way for network science applications.

6.2 PERFORMANCE EVALUATION

This section presents an experimental evaluation of the performance of NetworKit's algorithms. Our platform is a shared-memory server with 256 GB RAM and 2x8 Intel(R) Xeon(R) E5-2680 cores (32 threads due to hyperthreading) at 2.7 GHz.

6.2.1 *Benchmark*

Fig. 22 shows results of a benchmark of the most important analytics kernels in NetworKit. The algorithms were applied to a diverse set of 15 real-world networks in the size range from 16k to 260M edges, including web graphs, social networks, connectome data and internet topology networks (see Table 4 for a description). Kernels with quadratic running time (like Betweenness) were restricted to the subset of the 4 smallest networks. The box plots illustrate the range of processing rates achieved (dots are outliers). The benchmark illustrates that a set of efficient linear-time kernels, including ConnectedComponents, the community detectors, PageRank, CoreDecomposition and ClusteringCoefficient, scales well to networks in the order of 10^8 edges. The iFub [CGH+13] algorithm demonstrates its surprisingly good performance on complex networks, moving diameter calculation effectively into the class of linear-time kernels. Fig. 23 breaks its processing rate down to the particular instances, in decreasing order of size, illustrating that performance is often strongly dependent on the specific structure of complex networks. Algorithms like BFS and ConnectedComponents actually scan every edge at a rate of 10^7 to 10^8 edges per

name	type	n	m	source
fb-Caltech36	social (friendship)	769	16656	[TMP12]
PGPgiantcompo	social (trust)	10680	24316	Boguña et al. 2014
coAuthorsDBLP	coauthorship (science)	299067	977676	[BMS+14a]
fb-Texas84	social (friendship)	36371	1590655	[TMP12]
Foursquare	social (friendship)	639014	3214986	[ZL09]
Lastfm	social (friendship)	1193699	4519020	[ZL09]
wiki-Talk	social	2394385	4659565	[LK14]
Flickr	social (friendship)	639014	55899882	[ZL09]
in-2004	web	1382908	13591473	[BCSV04]
actor-collaboration	collaboration (film)	382219	15038083	[Kun13a]
eu-2005	web	862664	16138468	[BCSV04]
flickr-growth-u	social (friendship)	2302925	33140017	[Kun13a]
con-fiber_big	brain (connectivity)	591428	46374120	openconnecto.me
Twitter	social (followership)	15395404	85331845	[ZL09]
uk-2002	web	18520486	261787258	[BCSV04]
uk-2007-05	web	105896555	3301876564	[BCSV04]

Table 4: Networks used for the evaluation

second. Betweenness calculation remains very time-consuming in spite of parallelization, but approximate results can be obtained two orders of magnitudes faster.

6.2.2 *Comparative Benchmark*

NetworKit, igraph and graph-tool rely on the same hybrid architecture of C/C++ implementations with a Python interface. igraph uses non-parallel C code while graph-tool also features parallelism. We benchmarked typical analysis kernels for the three packages in comparison on the aforementioned parallel platform and present the measured performance in Fig. 24. Where applicable, algorithm parameters were selected to ensure a fair comparison. In this respect it should be mentioned that graph-tool's implementation of Brandes' betweenness algorithm does more work as it also calculates edge betweenness scores during the run. (Anyway, performance differences in the implementation quickly become irrelevant for a non-linear-time algorithm as the input size grows.) graph-tool also takes a different approach to community detection, hence the comparison is between igraph and NetworKit only. We summarize the benchmark results as follows: In our benchmark, NetworKit was the only framework that could consistently run the set of kernels (excluding the quadratic-time betweenness) on the full set of networks in the timeframe of an overnight run. For some of igraph's and graph-tool's implementations the test set had to be restricted to a subset of smaller networks to make it possible to run the complete benchmark overnight. NetworKit has the fastest average processing rate on all of these typical analytics kernels. Our implementations have a slight edge over the others for breadth-first search, connected components, clustering coefficients and betweenness. Considering that the algorithms are very similar, this is likely due to subtle differences and optimizations in the implementation. For PageRank, core decomposition and the

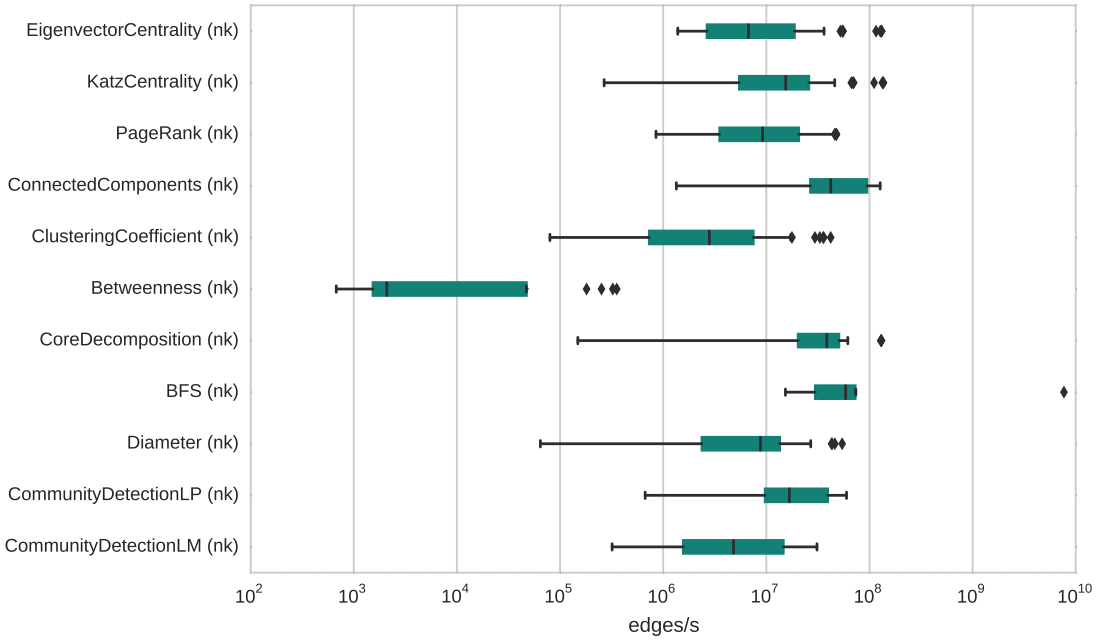


Figure 22: Processing rates of NetworKit analytics kernels

two community detection algorithms, our parallel methods lead to a larger speed advantage. The massive difference for the diameter calculation is due to our choice of the iFub algorithm [CGH+13], which has better running time in the typical case (i.e. complex networks with hub structure) and enables the processing of inputs that are orders of magnitudes larger.

Another scalability factor is the memory footprint of the graph data structure. NetworKit provides a lean implementation in which the 260M edges of the `uk-2002` web graph occupy only 9 GB, compared with igraph (93GB) and graph-tool (14GB). After indexing the edges, e.g. in order to compute edge centrality scores, NetworKit requires 11 GB for the graph.

A third factor that should not be ignored for real workflows is I/O. Getting a large graph from hard disk to memory often takes far longer than the actual analysis. For our benchmark, we chose the GML graph file format for the input files, because it is supported by all three frameworks. We observed that the NetworKit parser is significantly faster for these non-attributed graphs.

6.3 COMPARISON WITH DISTRIBUTED COMPUTING FRAMEWORKS

Many applications – think for example of the networks collected by online social networking sites – produce enormous amounts of network data that exceed the memory of any single computer, calling for distributed solutions. For instance, Facebook applies the Apache Giraph framework (discussed below) for the analysis of its social graph [Ave13]. A distributed system consists of a set of autonomous processors, each with their own memory, running software that performs computations and exchanges data as messages via a network. Distributed systems are highly scalable as computers can be added easily. Multiple programming models for distributed parallel computing have been introduced, some general-purpose (like MapReduce), others focus on graph processing (like Pregel).

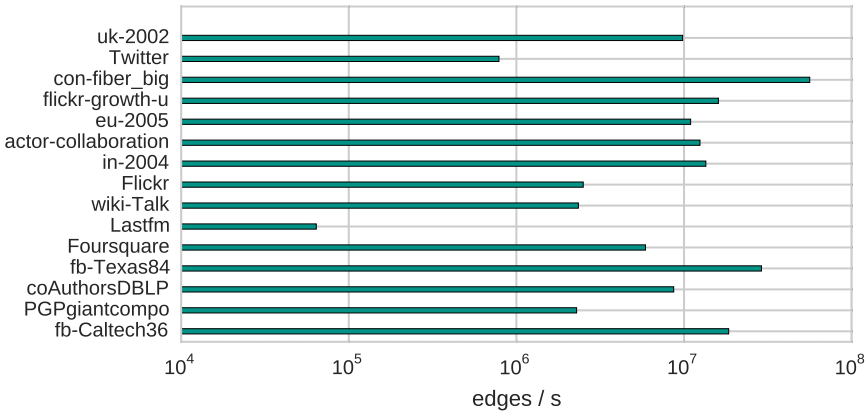


Figure 23: Processing rate of diameter calculation on different networks

Among the software frameworks implementing these programming models, we consider Apache Giraph [Apa15b], Apache Giraph++ [TBC+13], GraphLab [Dat15] and Apache Flink [Apa15a] in our study. Considering performance, the scalability and expressiveness enabled by such frameworks comes at the cost of overhead: A recent experimental study [SSP+14] compares implementations of graph algorithms based on frameworks (including GraphLab and Giraph) to ad hoc, hand-optimized code and observes a substantial performance gap. A similar new study [MIM15] asserts that for typical distributed computing frameworks often hundreds of computing nodes are needed to outperform a single-threaded implementation for common graph algorithms on graphs of a few billion edges, and raises the question to which extent their good scalability is due to merely distributing their own overheads. Moreover, complex networks present certain inherent performance challenges for distributed systems: Computation is data-driven in this case, i.e. it is largely determined by the graph structure. This makes it difficult to exploit opportunities for parallelism, because they depend heavily on the input. Real-world networks often have a skewed degree distribution, which adversely affects load balance. In synchronous execution models, the few high-degree vertices will delay the whole system. Performance depends also on how vertices are distributed among the processor nodes, i.e. the graph must be partitioned as uniformly as possible to reduce communication volume and to balance the work load – a problem whose optimal solution is $\mathcal{NP}$-hard to compute. Given these challenges, performance experiments on real-world networks are crucial for choosing the right framework for the problem.

In the context of distributed computing systems we briefly deviate from the previously defined terms and speak of "nodes" as the single computers making up a distributed system, and "vertices" as the nodes of a graph, to avoid confusion.

6.3.1 *Distributed Programming Models and Frameworks*

What follows is a brief description of the distributed computing frameworks considered for the experimental study. Each of them implements a specific distributed programming model.

MapReduce is a programming model for general distributed computation [DG08], and is often considered the de facto standard for this purpose [KSV10]. Implementations of

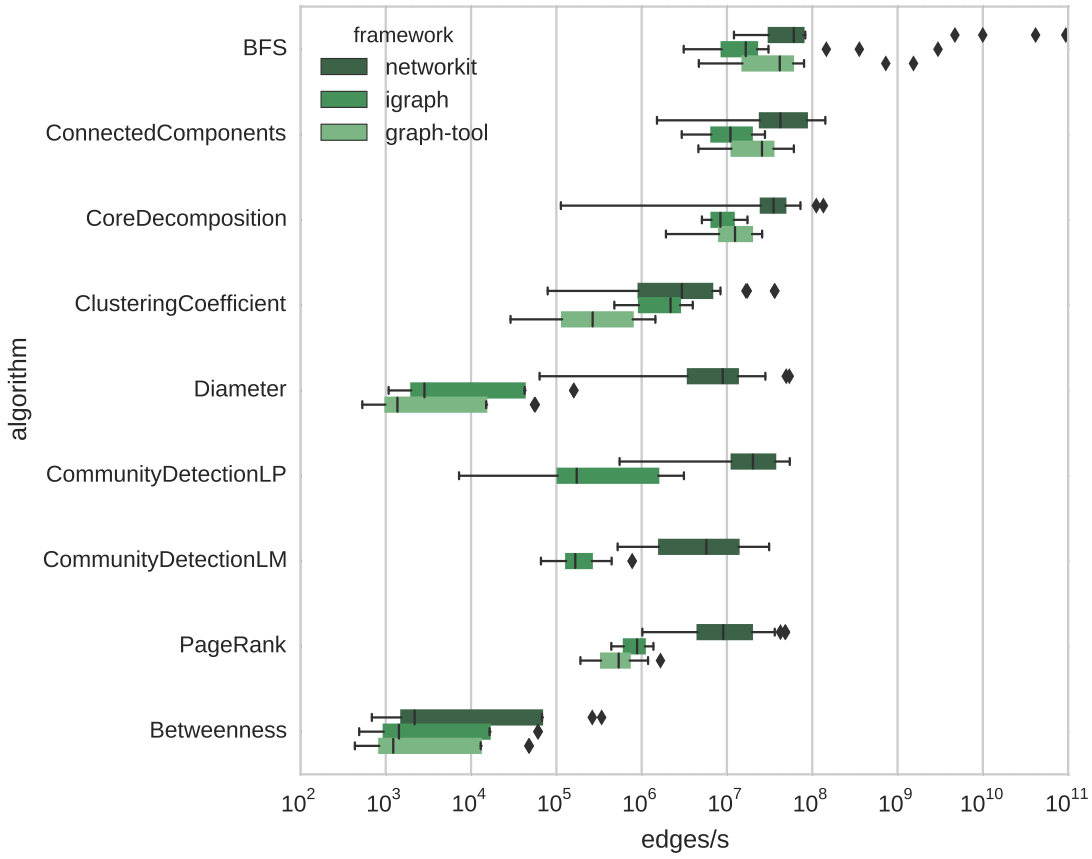


Figure 24: Processing rates of typical analytics tasks: NetworKit in comparison with igraph and graph-tool

the model are provided by frameworks such as *Hadoop*. Although MapReduce is able to express many common graph algorithms [LD10], it has limited practicality in this area: For instance, many graph analysis algorithms are iterative. Since the MapReduce model does not support iteration, this can only be reproduced by scheduling several consecutive jobs, resulting in significant overhead. Workarounds like emitting the graph structure make MapReduce implementations harder to develop and understand. We do not include MapReduce in the experimental study, since the models we discuss in the following offer more intuitive ways of expressing graph algorithms.

The PACT ("Parallelization Contract") model [BEH+10] was proposed to extend MapReduce in both functionality and optimization opportunities. It can be seen as a generalization of MapReduce that offers additional operators. When writing a PACT program, the programmer uses an operator by defining its user function and the input data it operates on. The PACT compiler examines the structure of the resulting operator graph and infers optimization strategy for the execution, e.g.rearranging operators in the graph. A PACT program therefore has a declarative style, reminiscient of the SQL language. The PACT model is implemented by the Apache Flink framework [Apa15a], written in Java.

Vertex-centric models express programs from a vertex-perspective. These programs are executed iteratively for each node in the graph, performing a computation updating the node's state based on data collected from neighboring graph nodes. The models are tailored to graph algorithms and by incorporating nodes and edges, they include the data

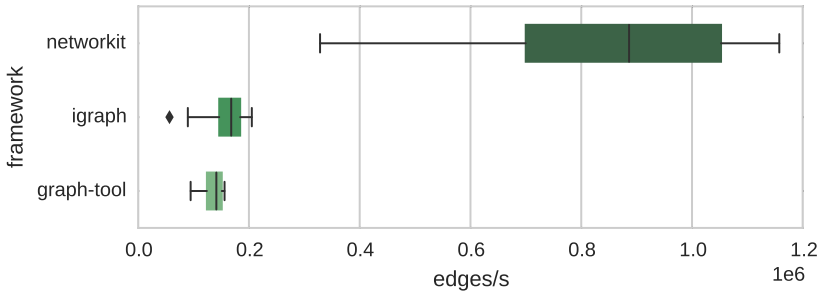


Figure 25: I/O rates of reading a graph from a GML file: NetworKit in comparison with igraph and graph-tool

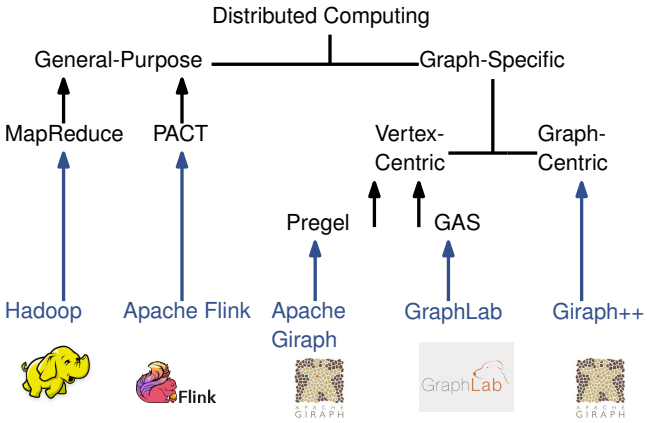


Figure 26: Overview and classification of distributed pro- gramming models (black) and implementations (blue) considered

dependencies of graphs in the programming model. Among vertex-centric models we can distinguish the conceptually similar Pregel and Gather-Apply-Scatter (GAS) models. In the Pregel model [MAB+10], an iteration consists of a vertex-centric algorithm that is executed on each processor, followed by a communication phase for data exchange between processors. The Pregel model is implemented by the Apache Giraph framework [Apa15b], in Java. GAS programs are iterative and vertex-centric like Pregel programs, but decompose an iteration into the gather, apply and scatter phases. Accordingly, a GAS program is defined by a `gather`, `apply` and `scatter` function. The `gather` and `scatter` functions only have read access to vertex data, which allows them to be executed concurrently without the need for synchronization. The GAS model is implemented by the GraphLab framework [Dat15], written in C++.

One of the reasons why vertex programs are so intuitive is that the abstraction hides details of the distributed execution, like graph partitioning. However, this information could be used for algorithm-specific optimizations. The graph-centric model [TBC+13] is a lower-level abstraction which provides information on the partition structure to the programmer: A program is no longer expressed for one vertex, but for the whole partition. In the graph-centric model, messages need to be sent to boundary vertices only, as we can directly update vertex states of internal vertices. This reduces the amount of messages that are sent and processed, possibly leading to faster convergence and execution time.

name	n	m	size
twitter-l	52M	1,963M	36.16
twitter	41M	1,468M	22.35
wikipedia-links-en	27M	601M	9.48
uk-2002	18M	261M	8.26
orkut-links	3M	117M	1.65
livejournal-links-d	4.8M	68M	0.99
livejournal-links-u	5M	49M	0.67
flickr-links	1.7M	15M	0.18
flickr-edges	105k	2.3M	0.02

Table 5: Properties of the networks (with n vertices and m edges) used in the experiments. Size denotes the size of the graph stored in edge list format in GB.

The graph-centric model is implemented by the Giraph++ [TBC+13] framework, an extension to Apache Giraph

6.3.2 *Experimental Setup*

For the experimental study, we select four common and typical network analysis problems, finding connected components and communities (cf. Sec. 2.4), and the calculation of PageRank and clustering coefficients (cf. Sec. 2.3). The former two call for a decomposition of the entire network into cohesive parts, and we opt for algorithms based on label propagation (cf. 8.2.1) which lend themselves to vertex-centric implementations. The latter two yield a ranking of vertices by structural importance, PageRank through a simple iterative rule, clustering coefficients based on the somewhat more intricate counting of triangles.

Real world network data sets used for the experiments, including a variety of web graphs and networks from online social media applications, are listed in Table 5.

For the experiments a cluster of 8 nodes was used. Each system has 24GB RAM and runs an Intel Xeon X5355 CPU with 2.66 GHz and 8 cores on two processors. The framework versions are GraphLab 2.2, Apache Giraph 1.1.0, Apache Flink 0.6 and NetworKit 3.3. NetworKit was run on a single node of the same type.

6.3.3 *Results*

The following plots show measured running times for the computation of connected components (Fig. 27), community detection (Fig. 28), PageRank (Fig. 29) and clustering coefficients (Fig. 30). We demonstrate that this distributed setting is able to process graphs of almost two billion edges, using standard commodity hardware. Not all frameworks perform equally well: Generally, GraphLab shows the best performance. Apache Giraph is less efficient in most cases, but offers an out-of-core mechanism which enables it to handle larger data sets. It is the only framework that could process the `twitter-l` graph. Giraph++ unexpectedly performs worse than the vertex-centric models. However, it must be remembered that only an experimental version of Giraph++, based on an outdated version of Apache Giraph, is available. Apache Flink performed worse than the vertex-centric frameworks. Also, larger graphs could not be processed for some of the

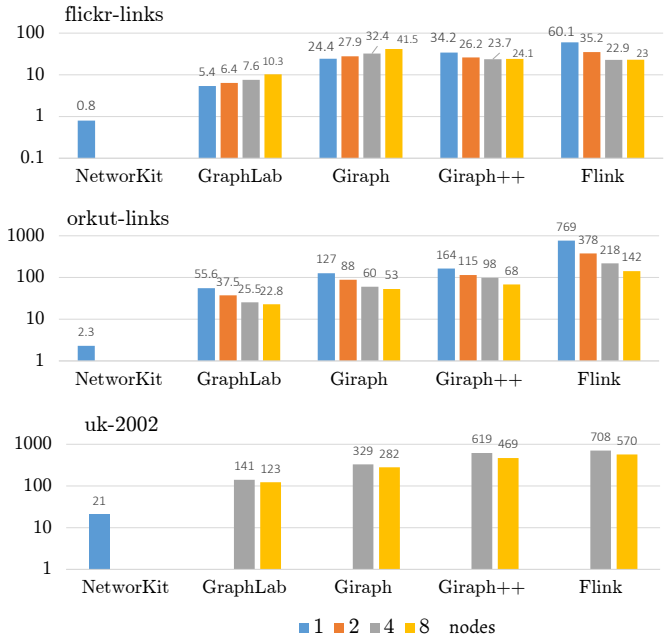


Figure 27: Running times [in seconds] of connected components computation.

algorithms due to issues with memory. Support for asynchronous execution is essential for the label propagation community detection algorithm, since the heuristic may not converge for synchronous execution due to label oscillation. The single-machine software package NetworKit outperforms the distributed frameworks in almost all scenarios. It can handle larger graphs than the distributed frameworks on a single machine, and its execution times are often only a fraction of the other frameworks', due to efficient native code and lack of overhead associated with the software machinery that enables and supports distributed computing. Consequently, the decision for a distributed computing solution for complex network analysis should come out of the necessity of massive graph data volumes that exhaust typical main memory capacity.

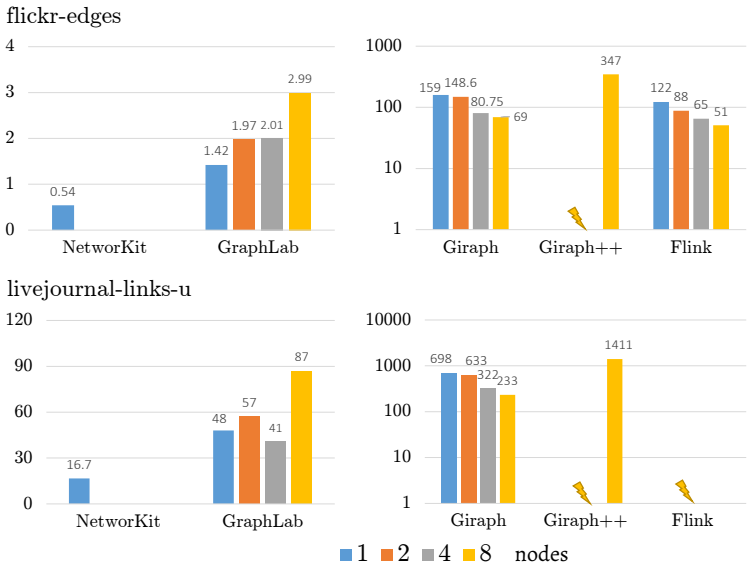


Figure 28: Running times [in seconds] of community detection on 1, 2, 4 and 8 nodes. The flash symbol indicates that a graph is too large to be processed using the given number of computing nodes.

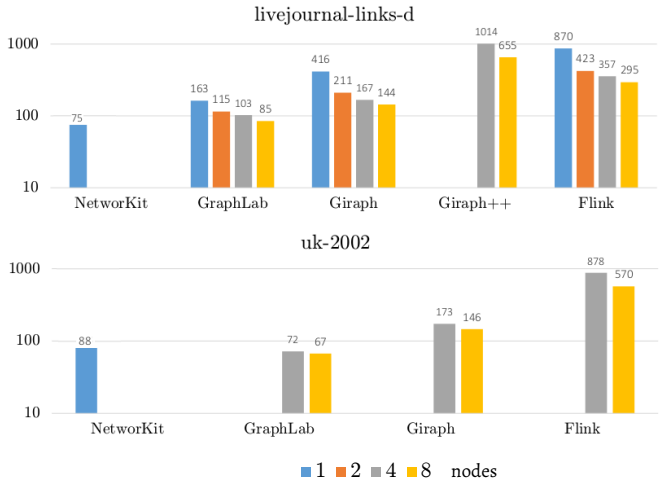


Figure 29: Running times [in seconds] of PageRank.

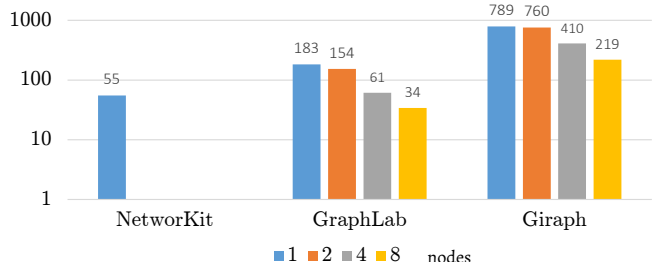


Figure 30: Running times [in seconds] of the clustering coefficient computation for the `flickr-edges` graph.

CONCLUSION OF PART II

The NetworKit project exists at the intersection of graph algorithm research and network science. Its contributors develop and collect state-of-the-art algorithms for network analysis tasks and incorporate them into ready-to-use software. The open-source package is under continuous development. The result is a tool suite of network analytics kernels, network generators and utility software to explore and characterize large network data sets on typical multicore computers. Part II detailed techniques that allow to scale to large networks, including appropriate algorithm patterns (parallelism, heuristics, data structures) and implementation patterns (e.g. modular design). The interface provided by our Python module allows domain experts to focus on data analysis workflows instead of the intricacies of programming. This is especially enabled by a new front end for explorative network analysis that generates comprehensive statistical reports on structural features of the network. Basic programming skills are required to operate NetworKit via its Python interface, but enables users to seamlessly integrate our toolkit into the Python ecosystem of data analysis tools. The hybrid architecture of core algorithms and data structures written in C++ and a Python front end, connected via the Cython tool chain, enables us to combine the best of both worlds in terms of performance and usability.

Among similar software packages, NetworKit yields the best performance for common analysis workflows. Our experimental study showed that NetworKit is capable of quickly processing large-scale networks for a variety of analytics kernels in a reliable manner. This translates into faster workflow and extended analysis capabilities in practice. We recommend NetworKit for the comprehensive structural analysis of large complex networks, as well as processing large batches of smaller networks. With fast parallel algorithms, scalability is in practice primarily limited by the size of the shared memory: A standard multicore workstation with 256 GB RAM can therefore process up to 10^{10} edge graphs. NetworKit also outperforms comparable implementations of graph algorithms in different distributed graph processing frameworks. It can handle larger graphs than the distributed frameworks on a single machine, and its execution times are often only a fraction of their time. This shows how significant performance advantages can be gained from tapping the full potential of shared-memory parallel programming for the analysis of large volumes of network data as long as they can still be fit into main memory.

With NetworKit we followed an open development model, publishing the source code under a minimally restrictive free software license and hosting it in a public repository, and incorporating code from a diverse set of contributors, including department colleagues, students and external collaborators. We also focused on product quality in terms of availability, practicality, usability, and documentation. This approach is not very common for software built in an academic setting, where software is frequently written by individuals or small groups only, and takes the form of prototypes built for generating data for publications. However, the experience of working on the NetworKit project has highlighted the benefits of this approach. Sharing a code base with others leads to a process of continuous peer review and is very helpful in building technical expertise. The efficiency gains we experienced from a shared software framework within our working groups, as well as the occasional positive feedback from users all over the world, came as

returns of the singificant time invested into project maintenance. An open development model also promotes good scientific practice, e.g. by facilitating reproduction of results.

Part III

COMMUNITY DETECTION IN COMPLEX NETWORKS

7 INTRODUCTION TO COMMUNITY DETECTION

> A student had created a clever pattern in Game of Life, and proudly showed it to Moore: "I can prove that it's behavior is undecidable, since it is equivalent to the Halting Problem". Moore ripped out the power cord from the computer, and the pattern vanished. "It has halted" he said. The student was enlightened, but the pattern was lost.
>
> – CS koan by John Grillo

Community detection in networks is the task of dividing a network into subgraphs which are internally densely and externally sparsely connected. Among manifold applications, community detection has been used to counteract search engine rank manipulation [Sch07], to discover scientific communities in publication databases [SSM+12], to identify functional groups of proteins in cancer research [JCZB06], and to organize content on social media sites [GLMY11]. This chapter discusses definitions, restrictions, formalizations, and methodic, scalable heuristics (Chapter 8).

7.1 MODULARITY

Extensive research on community detection in networks has given rise to a variety of definitions of what constitutes a good community and a variety of methods for finding such communities, many of which are described in surveys by Schaeffer [Sch07] and Fortunato [For10]. Among these definitions, the lowest common denominator is that a community is an internally dense node set with sparse connections to the rest of the graph. Areas of high edge density in a network can be treated as overlapping sets, and methods for detecting overlapping communities exist [PDFV05], but we restrict ourselves to finding disjoint communities, i.e. a partition (Def. 18) of the node set which uniquely assigns a node to a community. The quality measure *modularity* [GN02] formalizes the notion of a good partition into communities. For a given partition of the node set, modularity compares its *coverage* (fraction of edges within communities) to an expected value based on a random edge distribution model which preserves the degree distribution.

Definition 22 (Modularity)**.** For a graph $G = (V, E)$ and disjoint communities $\zeta = \{C_1, \ldots, C_k\}$ of G, *modularity* is defined as

$$M(G, \zeta) := \underbrace{\sum_{C \in \zeta} \frac{|E(C)|}{|E|}}_{\text{coverage}} - \underbrace{\sum_{C \in \zeta} \frac{\left(\sum_{v \in C} deg(v)\right)^2}{(2 \cdot |E|)^2}}_{\text{expected coverage}} \tag{15}$$

□

The specific null model was chosen in order to take the degree distribution of G into account, an important step since the node degrees of real complex networks deviate strongly from those of a simple (Erdős–Rényi) random graph. This null model effectively compares the coverage of the given partition on G with its coverage on a randomized

replica according to the Chung-Lu generative model (see Chapter 12). However, one might criticize singling out the degree distribution as the only property that should to be conserved, which opens up the possibility for other null models. Optimizing modularity is $\mathcal{NP}$-hard [BDG+08], but efficient heuristics will be discussed in the following. Modularity maximization has become a standard technique for community detection for several reasons: As a parameter-free method, modularity requires no prior knowledge about the network and therefore facilitates exploratory analysis. Its formulation makes it suitable for incremental maximization (as described in Chapter 8), leading to efficient algorithms. Not the least, the measure has been well-studied and often applied to real data – a somewhat circular but nonetheless relevant argument for its adoption.

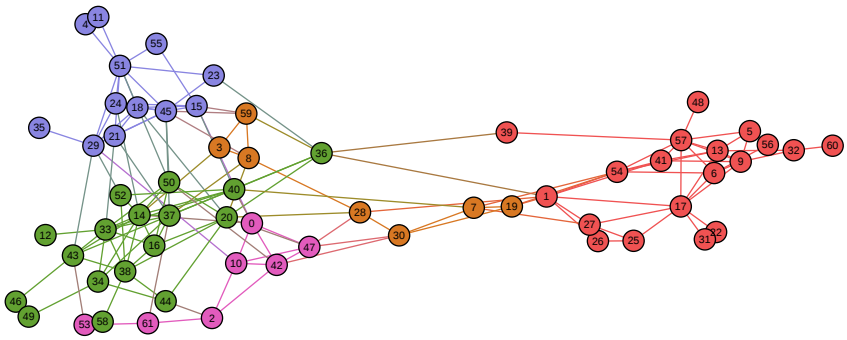


Figure 31: Node color represents modularity-driven communities

It is worth noting the following properties and behaviors of modularity [Gör10]:

- Isolated nodes have no impact on modularity.
- A partition with maximum modularity has no cluster that consists of a single node of degree 1. Hence such satellite nodes are always assigned to the community of their neighbor.
- A partition with maximum modularity does not include disconnected communities.
- Nonlocal behavior: Changes to the graph may affect the community membership of a node even though its direct neighborhood has not changed.
- Scaling behavior and resolution limit [FB07]: The size and structure of communities in the modularity-optimal solution depend on the number of edges in G, with a bias towards communities of unform size $\approx \sqrt{m}$. Consequently, scaling up the network may result in the merger of communities without their internal or external linkage having changed.

While we should be aware of these properties and any resulting, possibly counter-intuitive behaviors in applications, none of them disqualify modularity in comparison with alternative techniques. The frequently criticized resolution limit in particular can be remedied by applying a parametrized version of the measure, termed *multi-resolution modularity* [Lam10].

Definition 23 (Multi-Resolution Modularity)**.** For a graph $G = (V, E)$ and disjoint communities $\zeta = \{C_1, \ldots, C_k\}$ of G, *multi-resolution modularity* is defined as

$$M_\gamma(G, \zeta, \gamma) := \underbrace{\sum_{C \in \zeta} \frac{|E(C)|}{|E|}}_{\text{coverage}} - \gamma \cdot \underbrace{\sum_{C \in \zeta} \frac{\left(\sum_{v \in C} deg(v)\right)^2}{(2 \cdot |E|)^2}}_{\text{expected coverage}} \tag{16}$$

where γ is in the range $[0, 2m]$. □

Multiplying the expected coverage term with a parameter γ allows us to influence the resolution, $\gamma = 0$ yielding a single community, $\gamma = 1$ being standard modularity and $\gamma = 2m$ producing singleton communities. The approach reaches its limits in networks with a very heterogeneous community size distribution, since it is not possible to tune resolution to simultaneously avoid a bias towards large and towards small communities, a behavior that is conjectured to be a general problem of methods based on global optimization [LF11].

Another point worth discussing is whether modularity can meaningfully be treated and interpreted as a property of a given network, as occasionally seen in the literature. Its use as a measure of network structure is problematic in multiple ways: First of all, modularity is a function of a graph and a specific partition of the node set, and the latter cannot be omitted. A large variety of community detection algorithms (cf. Sec. 8.1) are in use, which may be nondeterministic or may arrive at different local optima depending on subtle implementation differences, so that reported modularity values may be difficult to reproduce. Now if the network science community were to standardize and agree on a single deterministic algorithm (which seems unlikely given the current state of the art), would the reported modularity values be meaningful indicators of the network? Does the achieved modularity allow conclusions about the "strength" or the nature of the community structure? Certainly it is not independent from the network's patterns of edge density and sparsity, but interpretation and comparison is very difficult, in part due to the above-mentioned properties. First and foremost, modularity is clearly not independent of network size, and comparison between networks of different scale becomes meaningless. In conclusion, it is sensible to limit the use of modularity to a tool for comparison among partitions of a given network, not for comparison of networks.

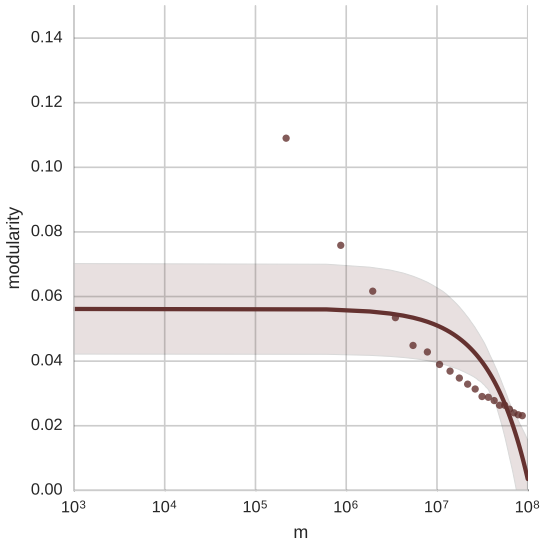


Figure 32: Erdős–Rényi random graphs do not have a significant community structure. Nonetheless modularity values of communities found by the PLM algorithm (see Chapter 8) decrease with the network size, while the (lack of) community structure essentially stays the same. Linear regression line and residuals shown.

Modularity is also geared towards the global community detection problem, assigning each node of the network to a community, and the problem of detecting communities in a part of the network only needs different approaches (cf. Chapter 9).

7.2 ALTERNATIVE APPROACHES

In the following we focus on modularity as a measure of how well a given partition captures the community structure of a network, and an objective function for explorative community detection. Modularity is well-tested, well-understood and forms the basis of several scalable heuristics. The evaluation in Chapter 8 uses modularity and is limited to a set of algorithms that implicitly or explicitly target modularity. However, this approach is not without alternatives, which should be briefly mentioned here. Recently, an alternative quality function named *surprise* has been introduced [AM11]. As advantages over modularity, the authors claim that surprise performs better in synthetic benchmarks (see Sec. 7.3) and does not have a resolution limit to the same extent as modularity. A more in-depth analysis of suprise is provided in the PhD thesis of Andrea Kappes, who shows that surprise optimization is $\mathcal{NP}$-hard as well, and clarifies experimentally that suprise-driven community detection yields significantly more and smaller clusters compared to modularity, and far more single nodes assigned to a community of one [Kap15]. The Infomap algorithm framework [RAB09] (`http://www.mapequation.org`) takes an approach based on information-theoretic encodings of random walks (represented as sequences of visited nodes). Because random walks tend to get trapped within communities, leading to redundant node sequences, the length of their encodings can be used to detect communities. Infomap does not exhibit the scaling behavior and resolution limit of modularity-maximization. Another family of methods is based on fitting a *stochastic block model* to the given network, i.e. partitioning the node set into "blocks" and estimat-

ing intra- and inter-block edge probabilities so that blocks capture communities [KN11] [DKMZ11].

7.3 EVALUATION OF COMMUNITY DETECTION METHODS

In addition to running time, community detection algorithms need to be evaluated with respect to accuracy. Ideally experiments would be based on a desired partition for the algorithm to detect, which would serve as *ground truth*. A suitable ground truth partition of a network is a partition which generates internal density and external sparsity of edges.

However, this is often out of reach when dealing with real-world network data. It is true that some of the network data sets frequently used for network algorithm engineering have associated "ground truth communities" data that assigns the node set into subsets [LK14] (e.g. an online social network with user-defined groups). The scare quotes are intentional: It is a problematic assumption that a community detection method is better if it finds these sets with higher accuracy. Given such a "ground truth", we need to ask whether it is exactly this partition that coincides with the pattern of edge density and sparsity that we want to detect. No evidence for this is provided in the case of [LK14]. While it is plausible some influence, it is unlikely that it will coincide with what modularity-maximization yields, or any formal, graph-based definition of communities. Community structure is a multi-factorial phenomenon in real networks, driven by multiple node attributes and other factors, rarely reducible to a single node attribute. Hence, for the vast majority of real-world networks, we lack a ground-truth partition that is reliable in the context of evaluating and tuning general community detection algorithms.

Consequently, to measure the accuracy of a method for algorithm engineering purposes, we need to rely on generative network models (which are discussed mainly in Chapter 12) to produce synthetic graphs with patterns of edge density and sparsity in a controlled manner. A suitable and established method used several times in this thesis (Sec. 8.4.7, Sec. 9.5.1) is the *LFR generator* [LF09a]. The generator produces graphs that resemble real complex networks and contain dense communities which are the more sparsely connected the lower the mixing parameter μ. Algorithm performance is measured as the accuracy in recognizing the ground truth communities supplied by the generator, in view of increasing difficulty.

In applications of community detection methods, one should look for further validation of the detected communities beyond good modularity. This might include questions on how the solution helps to formulate hypotheses about real-world phenomena on the basis of network data. Whether one solution is more appropriate than another may therefore strongly depend on the domain of the network. However, this kind of domain-specific evaluation goes beyond the scope of this thesis.

8

ENGINEERING PARALLEL ALGORITHMS FOR COMMUNITY DETECTION

> Being snowed in together through the winter months will strain even the closest of friendships; Yíwen and Hwídah were no exception to this rule.
>
> Hwídah was irritated enough when her lanky roommate began dancing silently around their quarters, swinging her arms and legs within inches of Hwídah's nose. But Hwídah's patience came to an end one day when Yíwen set up an electric guzheng in the center of the room and began playing it—or rather, began plucking its strings inexpertly to produce a series of dissonant, tuneless, tempoless sounds. The interlude lasted a minute, after which Yíwen sat down and scribbled on some papers. But then she rose and repeated the performance.
>
> After the tenth iteration, Hwídah hurled a sandal right into Yíwen's backside, causing the girl to yelp and turn around.
>
> "What," growled Hwídah, "are you doing?"
>
> Some of Yíwen's papers fluttered to the floor. Hwídah snatched them up. They were printouts of quicksort implemented in different languages: C, Lisp, Perl, even Prolog. Each was covered with musical notations in red ink.
>
> "A thousand pardons for my rudeness," said Yíwen. "I have been attempting to encode certain algorithms as movements through space, or notes in the air. If the result is not pleasing I change my encoding and try again."
>
> "Why?" asked Hwídah.
>
> "To see what I will discover by doing it," answered Yíwen. "We speak often of the beauty or elegance of code. Perhaps, without knowing it, we have been composing choreographies for information to dance to, and we find certain ones pleasing because they appeal to some deeper aesthetic sense common to other forms of human art. If so, then these arts would be connected. I seek that connection."
>
> Hwídah considered this.
>
> "Thus far, the music of quicksort eludes me," continued Yíwen with a sigh. "Perhaps my experiment is as foolish as translating songs into code and attempting to compile it. Perhaps the music of quicksort is best played by machines for their own appreciation, and not ours. Shall I quit this endeavor, and spare your nerves?"
>
> In answer, Hwídah produced two bits of cotton from her nightstand and put them in her ears. Yíwen bowed and returned to her instrument. Thus was peace restored.
>
> – from *The Codeless Code*

Despite extensive research on heuristics for community detection in networks, only few parallel codes exist, although parallelism will be necessary to scale to the data volume of real-world applications. This deficit in computing capability is addressed by a flexible and extensible community detection framework with shared-memory parallelism. Within this framework efficient parallel community detection heuristics are designed and implemented: A parallel label propagation scheme (PLP); the first large-scale parallelization of the well-known Louvain method, as well as an extension of the method adding refinement; and an ensemble scheme combining the above. In extensive experiments driven by the algorithm engineering paradigm the most successful parameters and combinations of these algorithms are identified, and compared with state-of-the-art competitors. The processing rate of the fastest algorithm (PLP) often reaches 50 M edges/second. The parallel Louvain method (PLM) and its variant with refinement is recommended as both quali-

tatively strong and fast. The presented methods prove highly scalable and are suitable for massive graphs with billions of edges.

This chapter is based on joint work with Henning Meyerhenke. Michael Hamann contributed additional optimizations for the PLM algorithm. Results were first presented at *International Conference on Parallel Processing 2013* in Lyon [SM13]. An extended paper appeared in the *IEEE Transactions on Parallel and Distributed Systems* [SM16].
The authors thank Pratistha Bhattarai for help with the experimental study, and Michael Hamann for optimizations to the implementation of the PLM algorithm.

8.1 STATE OF THE ART

This section gives a short overview over related efforts on building high-performance community detection algorithms and implementations. Among efficient heuristics for community detection we can distinguish between those based on community agglomeration and those based on local node moves. Agglomerative algorithms successively merge pairs of communities so that an improvement with respect to community quality is achieved. In contrast, local movers search for quality gains which can be achieved by moving a node to the community of a neighbor.

In order to clarify the state of the art, we treat the *10th DIMACS Implementation Challenge* [BMSW13] as an important benchmark. This scientific competition accepted submissions of implementations for the community detection problem, and compared them with regards to running time and achieved modularity on a wide set of graphs.

A globally greedy agglomerative method known as CNM [CNM04] runs in $O(md \log n)$ for graphs with n nodes and m edges, where d is the depth of the dendrogram of mergers and typically $d \sim \log n$. Among the few parallel implementations competing in the *DIMACS* challenge, Fagginger Auer and Bisseling [FB13] submitted an agglomerative algorithm with an implementation for both the GPU (using *NVIDIA CUDA*) and the CPU (using *Intel TBB*). The algorithm weights all edges with the difference in modularity resulting from a contraction of the edge, then computes a heavy matching M and contracts according to M. This process continues recursively with a hierarchy of successively smaller graphs. The matching procedure can adapt to star-like structures in the graph to avoid insufficient parallelism due to small matchings. In the challenge, the CPU implementation competed as CLU_TBB and proved exceptionally fast. Independently, Riedy et al. [RMEB13] developed a similar method, which follows the same principle but does not provide the adaptation to star-like structures. An improved implementation, labeled CEL in the following, corresponds to the description in [RB13].

Community detection by label propagation belongs to the class of local move heuristics. It has originally been described by Raghavan et al. [RAK07]. Several variants of the algorithm exist, one of them (under the name *peer pressure clustering*) is due to Gilbert et al. [GRS07]. The latter use the algorithm as a prototype application within a parallel toolbox that uses numerical algorithms for combinatorial problems. Unfortunately, Gilbert et al. report running times only for a different algorithm, which solves a very specific benchmark problem and is not applicable in our context. A variant of label propagation by Soman and Narang [SN11] for multicore and GPU architectures exists, which seeks to improve quality by re-weighting the graph.

A locally greedy multilevel-algorithm known as the *Louvain method* [BGLL08] combines local node moves with a bottom-up multilevel approach. Bhowmick and Srinivasan

[BS13] presented a previous parallel version of the algorithm. According to their experimental results, our implementation is about four orders of magnitude faster. Noack and Rotta [RN11] evaluate similar sequential multilevel algorithms, which combine agglomeration with refinement.

Ovelgönne and Geyer-Schulz [OGS13] apply the *ensemble learning* paradigm to community detection. They develop what they call the *Core Groups Graph Clusterer* scheme, which we adapt as the *Ensemble Preprocessing* (EPP) algorithm. They also introduce an iterated scheme in which the core communities are again assigned to an ensemble, creating a hierarchy of solutions/coarsened graphs until quality does not improve any more. Within this framework, they employ *Randomized Greedy* (RG), a variant of the aforementioned CNM algorithm. It avoids a loss in solution quality that arises from highly unbalanced community sizes. The resulting CGGC algorithm emerged as the winner of the Pareto part of the DIMACS challenge, which related quality to speed according to specific rules. Recently Ovelgönne [Ove13] presented a distributed implementation (based on the big data framework *Hadoop*) of an ensemble preprocessing scheme using label propagation as a base algorithm. This implementation processes a 3.3 billion edge web graph in a few hours on a 50 machine Hadoop cluster [Ove13]. (Our *OpenMP*-based implementation of the similar EPP algorithm requires only 4 *minutes* on a shared-memory machine with 16 physical cores.)

From an algorithmic perspective, disjoint community detection is related to graph partitioning (GP). Although the problems are different in important aspects (unbalanced vs balanced blocks, unknown vs known number of blocks, different objectives), algorithms such as the Louvain method or PLMR bear conceptual resemblance to multilevel graph partitioners. Exploiting parallelism has been studied extensively for GP. Several established tools are discussed in recent surveys [BS11] [BMS+14b], most of them for machines with distributed memory. Often employed techniques are parallel matchings for coarsening and parallel variants of Fiduccia-Mattheyses for local improvement. These techniques are at best partially helpful in our scenario since vanilla matching-based coarsening is ineffective on complex networks and distributed-memory parallelism is not necessary for us.

Apart from the *10th DIMACS Implementation Challenge*, there are several more recent related efforts. A study on multithreaded GP by LaSalle and Karypis [LK13] explores the design space of multithreaded GP algorithms. Their results provide interesting insights, but are not completely transferable to our scenario. Following publication of our work, they presented Nerstrand [LK15], a fast parallel community detection algorithm based on modularity maximization and the multilevel paradigm, using different aggregation schemes. They use a similar experimental setup, also based on the *10th DIMACS Implementation Challenge* benchmark set, and report that Nerstrand yields modularity values comparable to PLMR several times faster on a subset of graphs of the benchmark set. However, they compare against an earlier version of the PLMR implementation that does not incorporate the latest performance improvements (described in Sec. 8.2.2). Considering that they yielded a speedup of about factor 2, Nerstrand still seems to have an edge over the current PLM implementation, though by a smaller margin. They find the PLP implementation to be faster than Nerstrand, albeit yielding lower modularity.

We observe that most efficient disjoint community detection heuristics make use of agglomeration or local node moves, possibly in combination with multilevel or ensemble techniques. Both basic approaches can be adapted for parallelism, but this is currently

the exception rather than the norm in our scenario. In this work we compare our own algorithms with the best currently available, sequential and parallel alike.

8.2 ALGORITHMS

8.2.1 *Parallel Label Propagation (PLP)*

Algorithm: Community detection by label propagation, as originally introduced by Raghavan et al. [RAK07], extracts communities from a labelling $V \to \mathbb{N}$ of the node set. Initially, each node is assigned a unique label, and then multiple iterations over the node set are performed: In each iteration, every node adopts the most frequent label in its neighborhood (breaking ties arbitrarily). Densely connected groups of nodes thus agree on a common label, and eventually a globally stable consensus is reached, which usually corresponds to a good solution for the network. Label propagation therefore finds communities in nearly linear time: Each iteration takes $O(m)$ time, and the algorithm has been empirically shown to reach a stable solution in only a few iterations, though not mathematically proven to do so. The number of iterations seems to depend more on the graph structure than the size. More theoretical analysis is done by [KPS13]. The algorithm can be described as a locally greedy *coverage* maximizer, i.e. it tries to maximize the fraction of edges which are placed within communities rather than across. With its purely local update rule, it tends to get stuck in local optima of *coverage* which implicitly are good solutions with respect to modularity: A label is likely to propagate through and cover a dense community, but unlikely to spread beyond bottlenecks. The local update rule and the absence of global variables make label propagation well-suited for a parallel implementation.

Algorithm 1 denotes PLP, our parallel variant of label propagation. We adapt the algorithm in a straightforward way to make it applicable to weighted graphs. Instead of the most frequent label, the *dominant label* in the neighborhood is chosen, i.e. the label l that maximizes $\sum_{u \in N(v):\zeta(u)=l} \omega(v,u)$. We continue the iteration until the number of nodes which changed their labels falls below a threshold θ.

Implementation: We make a few modifications to the original algorithm. In the original description [RAK07], nodes are traversed in random order. Since the cost of explicitly randomizing the node order in parallel is not insignificant, we make this optional and rely on some randomization through parallelism otherwise. We also observe that forgoing randomization has a negligible effect on quality. We avoid unnecessary computation by distinguishing between active and inactive nodes. It is unnecessary to recompute the label weights for a node whose neighborhood labels have not changed in the previous iteration. Nodes which already have the heaviest label become inactive (Algorithm 1, line 14), and are only reactivated if a neighboring node is updated (line 12). We restrict iteration to the set of active nodes. Iterations are repeated until the number of nodes updated falls below a threshold value. The motivation for setting threshold values other than zero is that on some graph instances, the majority of iterations are spent on updating only a very small fraction of high-degree nodes (see Figure 33 for an example). Since preliminary experiments have shown that time can be saved and quality is not significantly degraded by simply omitting these iterations, we set an update threshold of $\theta = n \cdot 10^{-5}$. Note that we do not use the termination criterion specified in [OGS13] as it does not lead to

Algorithm 1: PLP: Parallel Label Propagation

```
Input: graph G = (V, E)
Result: communities ζ : V → ℕ
parallel for v ∈ V
    ζ(v) ← id(v)
updated ← n
V_active ← V
while updated > θ do
    updated ← 0
    parallel for v ∈ {u ∈ V_active : deg(u) > 0}
        l* ← arg max_l { Σ_{u∈N(v):ζ(u)=l} ω(v, u) }
        if ζ(v) ≠ l* then
            ζ(v) ← l*
            updated ← updated + 1
            V_active ← V_active ∪ N(v)
        else
            V_active ← V_active \ {v}
return ζ
```

convergence on some inputs. The original criterion is to stop when all nodes have the label of the relative majority in their neighborhood [RAK07].

Label propagation can be parallelized easily by dividing the range of nodes among multiple threads which operate on a common label array. This parallelization is not free of race conditions, since by the time the neighborhood of a node u is evaluated in iteration i to set $\zeta_i(u)$, a neighbor v might still have the previous iteration's label $\zeta_{i-1}(v)$ or already $\zeta_i(v)$. The outcome thus depends on the order of threads. However, these race conditions are acceptable and even beneficial in an ensemble setting since they introduce random variations and increase base solution diversity. This also corresponds to *asynchronous updating*, which has been found to avoid oscillation of labels on bipartite structures [RAK07]. When dealing with scale-free networks whose degree distribution follows a power law, assigning node ranges of equal size to each thread will lead to load imbalance as computational cost depends on the node degree. Instead of statically dividing the iteration among the threads, guided scheduling (with `#pragma omp parallel for schedule(guided)`) assigns node ranges of decreasing size from a queue to available threads. This way it can help to overcome load balancing issues, since threads processing large neighborhoods will receive fewer vertices in later phases of the dynamical assignment process. This introduces some overhead, but we observed that guided scheduling is generally superior to statically parallelized loops for algorithms iterating over graph adjacencies with heterogeneous degree distributions.

8.2.2 *Parallel Louvain Method (PLM)*

Algorithm: The *Louvain method* for community detection was first presented by [BGLL08]. It can be classified as a locally greedy, bottom-up multilevel algorithm and uses modularity as the objective function. In each pass, nodes are repeatedly moved to neighboring communities such that the locally maximal increase in modularity is achieved, until the

communities are stable. Algorithm 2 denotes this move phase. Then, the graph is coarsened according to the solution (by contracting each community into a supernode) and the procedure continues recursively, forming communities of communities. Finally, the communities in the coarsest graph determine those in the input graph by direct prolongation.

Computation of the objective function modularity is a central part of the algorithm. Let $\omega(u, C) := \sum_{\{u,v\}:v\in C} \omega(u, v)$ be the weight of all edges from u to nodes in community C, and define the *volume* of a node and a community as $vol(u) := \sum_{\{u,v\}:v\in N(u)}\omega(u, v) + 2 \cdot \omega(u, u)$ and $vol(C) := \sum_{u\in C} vol(u)$, respectively. Recall that the modularity of a solution is defined as in Eq.17. (The computation of multi-resolution modularity works analogously, see Section 7.1 for details).

$$mod(\zeta, G) := \sum_{C\in\zeta} \left(\frac{\omega(C)}{\omega(E)} - \frac{vol(C)^2}{4\omega(E)^2} \right) \tag{17}$$

Note that the change in modularity resulting from a node move can be calculated by scanning only the local neighborhood of the node, because the difference in modularity when moving node $u \in C$ to community D is:

$$\begin{aligned} \Delta mod(u,\ C \to D) = &\frac{\omega(u, D \setminus \{u\}) - \omega(u, C \setminus \{u\})}{\omega(E)} \\ &+ \frac{(vol(C \setminus \{u\}) - vol(D \setminus \{u\})) \cdot vol(u)}{2 \cdot \omega(E)^2} \end{aligned} \tag{18}$$

We introduce a shared-memory parallelization of the Louvain method (PLM, Algorithm 3) in which node moves are evaluated and performed in parallel instead of sequentially. This approach may work on stale data so that a monotonous modularity increase is no longer guaranteed. Suppose that during the evaluation of a possible move of node u other threads might have performed moves that affect the Δmod scores of u. In some cases this can lead to a move of u that actually decreases modularity. Still, such undesirable decisions can also be corrected in a following iteration, which is why the solution quality is not necessarily worse. Working only on independent sets of vertices in parallel does not provide a solution since the sets would have to be very small, limiting parallelism and/or leading to the undesirable effect of a very deep coarsening hierarchy. Concerns about termination turned out to be theoretical for our set of benchmark graphs, all of which can be successfully processed with PLM. The community size resolution produced by PLM can be varied through a parameter γ in the range $[0, 2m]$, 0 yielding a single community, 1 being standard modularity and $2m$ producing singletons. Tuning this parameter is a possible practical remedy [Lam10] against modularity's resolution limit.

Implementation: The main idea of (Algorithm 3) is to parallelize both the node move phase and the coarsening phase of the Louvain method. Since the computation of the Δmod scores is the most frequent operation, it needs to be very fast. We store and update some interim values, which is not apparent from the high-level pseudocode in Algorithm 3. An earlier implementation associated with each node a map in which the edge weight to neighboring communities was stored and updated when node moves occurred. A lock for each vertex v protected all read and write accesses to v's map since `std::map` is not thread-safe. Meant to avoid redundant computation, we later discovered that this introduces too much overhead (map operations, locks). Recomputing the

Algorithm 2: move: Local node moves for modularity gain

```
Input: graph G = (V, E), communities ζ : V → ℕ
Result: communities ζ : V → ℕ
repeat
    parallel for u ∈ V
        δ ← max_{v∈N(u)} {Δmod(u, ζ(u) → ζ(v))}
        C ← ζ(arg max_{v∈N(u)} {Δmod(u, ζ(u) → ζ(v))})
        if δ > 0 then
            ζ(u) ← C
until ζ stable
return ζ
```

Algorithm 3: PLM: Parallel Louvain Method

```
Input: graph G = (V, E)
Result: communities ζ : V → ℕ
ζ ← ζ_singleton(G)
ζ ← move(G, ζ)
if ζ changed then
    [G′, π] ← coarsen(G, ζ)
    ζ′ ← PLM(G′)
    ζ ← prolong(ζ′, G, G′, π)
return ζ
```

weight to neighbor communities each time a node is evaluated turned out to be faster. The current implementation only stores and updates the volume of each community. An additional optimization to the implementation eliminated the overhead associated with using an `std::map` to store for each node the weights of edges leading to neighboring communities. The mechanism was replaced by one `std::vector` for each of the p threads, leading to an acceleration of a factor of 2 on average, at the cost of a memory overhead of $O(p \cdot n)$. This implementation technique saves computational cost for the construction of the red-black tree data structure of `std::map`, and has been used before for the original implementation of the Louvain method. The former version (referred to as PLM*) can still be used optionally under tighter memory constraints.

Graph coarsening according to communities is performed in a straightforward way such that the nodes of a community in G are aggregated to a single node in G'. An edge between two nodes in G' receives as weight the sum of weights of inter-community edges in G, while self-loops receive the weight of intra-community edges. A mapping π of nodes in the fine graph to nodes in the coarse graph is also returned. In earlier versions of PLM, the graph coarsening phase proved to be a major sequential bottleneck. We address this problem with a parallel coarsening scheme: Each thread first scans a portion of the edges in G and constructs a coarse graph G'_t of its own. These partial graphs are then combined into G' by processing each node of G' in parallel and merging the adjacencies stored in each G'_t.

8.2.3 *Parallel Louvain Method with Refinement (PLMR)*

Following up on the work by Noack and Rotta on multilevel techniques and refinement heuristics [RN11], we extend the Louvain method by an additional move phase after each prolongation. This makes it possible to re-evaluate node assignments in view of the changes that happened on the next coarser level, giving additional opportunities for modularity improvement at the cost of additional iterations over the node set in each level of the hierarchy. We denote the method and implementation as PLMR for *Parallel Louvain Method with Refinement*. We present a recursive implementation in Algorithm 4 which uses the same concepts as PLM.

Algorithm 4: PLMR: Parallel Louvain Method with Refinement

```
Input: graph G = (V, E)
Result: communities ζ : V → ℕ
ζ ← ζ_singleton(G)
ζ ← move(ζ, G)
if ζ changed then
    [G', π] ← coarsen(G, ζ)
    ζ' ← PLMR(G')
    ζ ← prolong(ζ', G, G', π)
    ζ ← move(ζ, G)
return ζ
```

8.2.4 *Ensemble Preprocessing (EPP)*

In machine learning, *ensemble learning* is a strategy in which multiple *base classifiers* or *weak classifiers* are combined to form a strong classifier. Classification in this context can be understood as deciding whether a pair of nodes should belong to the same community. We follow this general idea, which has been applied successfully to graph clustering before [OGS13]. Subsequently, we describe an ensemble techniques EPP. We also briefly describe algorithms for combining multiple base solutions.

Algorithm 5: EPP: *Ensemble Preprocessing*

```
Input: graph G = (V, E), ensemble size b
Result: communities ζ : V → ℕ
parallel for i ∈ [1, b]
    ζ_i ← Base_i(G)
ζ̄ ← combine(ζ_1, ..., ζ_b)
G', π ← coarsen(G, ζ̄)
ζ' ← Final(G')
ζ ← prolong(ζ', G, G', π)
return ζ
```

In a preprocessing step, assign G to an ensemble of base algorithms. The graph is then coarsened according to the *core communities* $\bar{\zeta}$, which represent the consensus of the base algorithms. Coarsening reduces the problem size considerably, and implicitly identifies the contested and the unambiguous parts of the graph. After the preprocessing

phase, the coarsened graph G' is assigned to the final algorithm, whose result is applied to the input graph by prolongation. Our implementation of the ensemble technique EPP is agnostic to the base and final algorithms and can be instantiated with a variety of such algorithms. We instantiate the scheme with PLP as a base algorithm and PLMR as the final algorithm. Thus we achieve massive nested parallelism with several parallel PLP instances running concurrently in the first phase, and proceed in the second phase with the more expensive but qualitatively superior PLMR. This constitutes the EPP algorithm (Algorithm 5). We write EPP(b, Base, Final) to indicate the size of the ensemble b and the types of base and final algorithm.

Implementation: A consensus of $b > 1$ base algorithms is formed by combining the base solutions ζ_i in the following way: Only if a pair of nodes is classified as belonging to the same community in every ζ_i, then it is assigned to the same community in the core communities $\bar{\zeta}$. Formally, for all node pairs $u, v \in V$:

$$\forall i \in [1, b]\ \zeta_i(u) = \zeta_i(v) \quad \iff \quad \bar{\zeta}(u) = \bar{\zeta}(v). \tag{19}$$

We introduce a highly parallel combination algorithm based on *hashing*. With a suitable hash function $h(\zeta_1(v), \ldots, \zeta_b(v))$, the community identifiers of the base solutions are mapped to a new identifier $\bar{\zeta}(v)$ in the core communities. Except for unlikely hash collisions, a pair of nodes will be assigned to the same community only if the criterion above is satisfied. We use a relatively simple function called djb2 due to Bernstein,% [1] which appears sufficient for our purposes. The use of a b-way hash function is fast due to a high degree of parallelism.

8.3 EXPERIMENTAL SETUP

8.3.1 *Framework and Settings*

All implementations are based on NetworKit (see ii). Parallelism is achieved in the form of loop parallelization with *OpenMP*, using the `parallel for` directive with `schedule(guided)` where appropriate for improved load balancing.

For representative experiments we average quality and speed values over multiple runs in order to compensate for fluctuations. Table 6 provides information on the multicore platform used for all experiments.

	phipute1.iti.kit.edu
compiler	gcc 4.8.1
CPU	2 x 8 Cores: Intel(R) Xeon(R) E5-2680 0 @ 2.70GHz, 32 threads
RAM	256 GB
OS	SUSE 13.1-64

Table 6: Platform for experiments

1 hash functions: http://www.cse.yorku.ca/~oz/hash.html

8.3.2 *Network Data Sets*

We perform experiments on a variety of graphs from different categories of real-world and synthetic data sets. Our focus is on real-world complex networks, but to add variety some non-complex and synthetic instances are included as well. The test set includes web graphs (`uk-2002`, `eu-2005`, `in-2004`, `web-BerkStan`), internet topology networks (`as-22july06`, `as-Skitter`, `caidaRouterLevel`), social networks (`soc-LiveJournal`, `fb-Texas84`, `com-youtube`, `wiki-Talk`, `soc-pokec`, `com-orkut`), scientific coauthorship networks (`coAuthorsCiteseer`, `coPapersDBLP`), a connectome graph (`con-fiber_big`), a street network (`europe-osm`) and synthetic graphs (`G_n_pin_pout`, `kron_g500-...`, `hyperbolic-268M`). Therefore, we cover a range of graph-structural properties. Real-world complex networks are heterogeneous data sets, which makes it impossible to pick an ideal or generic instance from which to generalize. Our main test set is chosen such that it can be handled by competing codes as well. It contains 20 networks from different domains. With this test set we aim for generalizable results. Note that the achievable modularity for a network depends on its size and inherent community structure, which may or may not be distinctive, and varies widely among the instances. The majority of test networks are taken from the collection compiled for the *10th DIMACS Implementation Challenge*[2] as well as the *Stanford Large Network Dataset Collection*[3] and are freely available on the web. They are undirected, unweighted graphs. Table 7 gives an overview over graph sizes as well as some structural features: A high maximum node degree (maxdeg) indicates possible load balancing issues. The number of connected components (comp) points to isolated single nodes or small groups of nodes. A high average local clustering coefficient (lcc) is an indicator for the presence of dense subgraphs. We evaluate solution quality and running time for all of our own algorithms as well as several relevant competitors on this set. For those algorithms that can process in reasonable time the largest real-world graph available to us, a web graph of the .uk domain with $m \approx 3.3 \cdot 10^9$, we add further experiments (see Section 8.4.8). To measure strong scaling, we run our parallel algorithms on this web graph.

8.4 EXPERIMENTS AND RESULTS

In this section we report on a representative subset of our experimental results for our different parallel algorithms, as well as competing codes. Figures 43 and 44 (as well as Figures 35 and 36) show running time and quality differences broken down by the networks of our test set. The bars of the charts are in ascending order of graph size. We have selected a diverse test set and show results for each network. The Pareto evaluation (Section 8.4.2) then aims to condense this into a single performance score.

8.4.1 *Parallel Label Propagation (PLP)*

PLP is extremely fast and able to handle the large graphs easily. The "weak classifier" PLP is nonetheless able to detect an inherent community structure and produce a solution with reasonable modularity values, although it cannot distinguish communities in a Kronecker graph, which has a very weak community structure. To demonstrate strong

2 *DIMACS* collection: http://www.cc.gatech.edu/dimacs10/downloads.shtml
3 *Stanford* collection: http://snap.stanford.edu/data/index.html

network	n	m	maxdeg	comp	lcc
as-22july06	22963	48436	2390	1	0.3493
G_n_pin_pout	100000	501198	25	6	0.0040
caidaRouterLevel	192244	609066	1071	308	0.2016
coAuthorsCiteseer	227320	814134	1372	1	0.7629
fb-Texas84	36371	1590655	6312	4	0.1985
com-youtube	1157828	2987624	28754	22939	0.1725
wiki-Talk	2394385	4659565	100029	2555	0.1991
web-BerkStan	685231	6649470	84230	677	0.6343
as-Skitter	1696415	11095298	35455	756	0.2930
in-2004	1382908	13591473	21869	134	0.7013
coPapersDBLP	540486	15245729	3299	1	0.8111
eu-2005	862664	16138468	68963	1	0.6509
soc-pokec	1632804	22301964	14854	2	0.1223
soc-LiveJournal	4847571	43369619	20334	1876	0.3667
kron_g500-simple...	1048576	44619402	131503	253380	0.2096
con-fiber_big	591428	46374120	5166	727	0.6024
europe-osm	50912018	54054660	13	1	0.0012
com-orkut	3072627	117185083	33313	187	0.1735
uk-2002	18520486	261787258	194955	38359	0.6892
hyperbolic-268M	6710886	268851810	71585	1	0.7895
uk-2007-05	105896555	3301876564	975419	756936	0.743

Table 7: Overview of networks used in experiments

scaling behavior, we apply PLP to the large uk-2007-05 web graph and increase the number of threads from 1 to 32 (Figure 38). (Weak scaling results on PLP and PLM are shown in Figure 48.) A speedup of about factor 8 is achieved when scaling from 1 to 32 threads. Note that we have only 16 physical cores and the step from 16 to 32 threads implies hyperthreading, so that a lower speedup is expected. Our results indicate that PLP can benefit from increased parallelism. Figure 34 breaks running times down by iteration, showing that the vast majority of time is spent in the first couple of iterations.

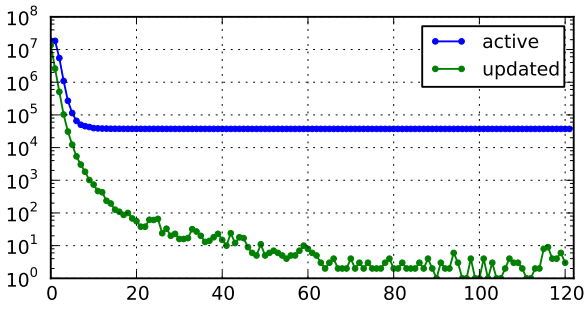


Figure 33: Number of active and updated labels per iteration of PLP for the web graph uk-2002.

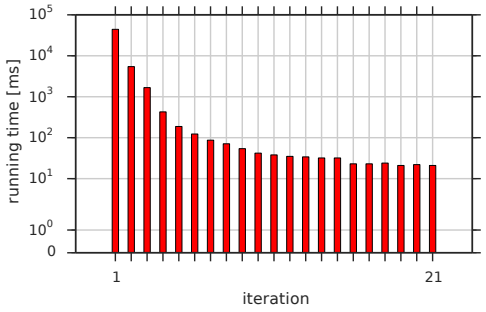
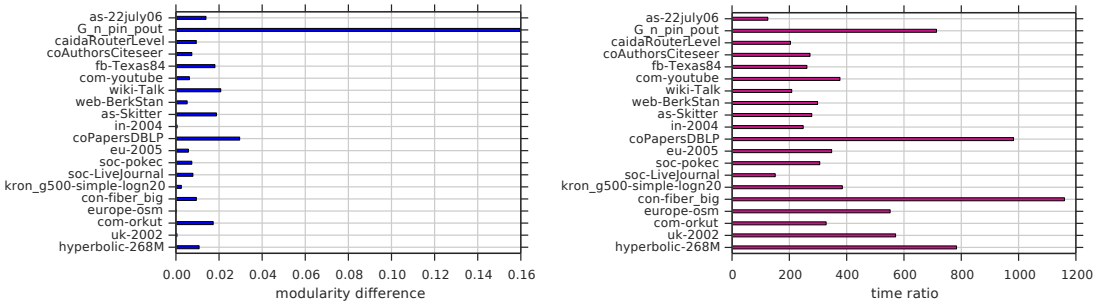


Figure 34: PLP running time in milliseconds per iteration for the `uk-2007-05` web graph, at 32 threads.

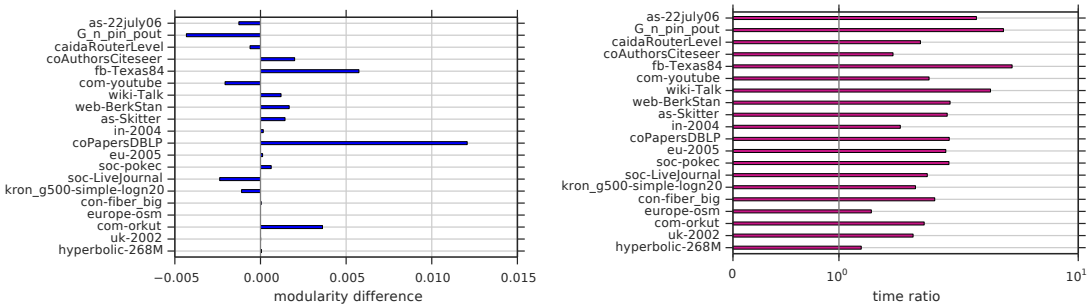


(a) CGGCi

Figure 35: Performance of the competitor algorithm CGGCi relative to baseline PLM.

8.4.2 *Pareto Evaluation*

Observed effects may vary strongly from one network to another, a sign of the heterogeneity of real-world complex networks. In addition to breaking down results by data set in the following, we want to give a condensed picture of the results. For this purpose we use the previous experimental data to compute a score for running time and solution quality. The time score is the geometric mean of running time ratios over our test set of networks with the running time of PLM as the baseline, while the modularity score is the arithmetic mean of absolute modularity differences. Figure 39 shows the resulting points. It becomes clear that all algorithms except CEL and EPP are placed on or close to the Pareto frontier. PLP is unrivaled in terms of time to solution, but solution quality is suboptimal. In the middle ground between label propagation and Louvain method, the parallel CLU_TBB achieves about the same modularity but beats the ensemble approach in terms of speed. PLM and PLMR emerge as qualitatively strong and fast candidates, closest to the lower right corner. (Their more memory-efficient implementation PLM* is about a factor of 2 slower.) It is also evident that our extended version PLMR can improve solution quality for a small computational extra charge. We recommend both PLM and PLMR as the default algorithms for parallel community detection in large networks. The original sequential implementation of the Louvain method is thus no longer on the Pareto frontier since it cannot benefit from multicore systems. RG and its ensemble combinations have the best modularity scores by a narrow margin, while they are by far



(a) PLM*

Figure 36: Performance of our PLM* algorithm relative to baseline PLM.

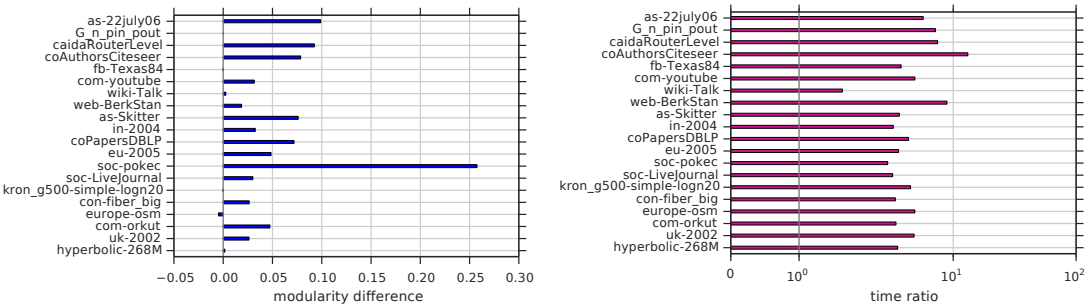


Figure 37: Difference in quality (left) and running time time ratio (right) of EPP(4,PLP,PLMR) compared to a single PLP.

the most computationally expensive ones, which places them outside of the application scenario we target.

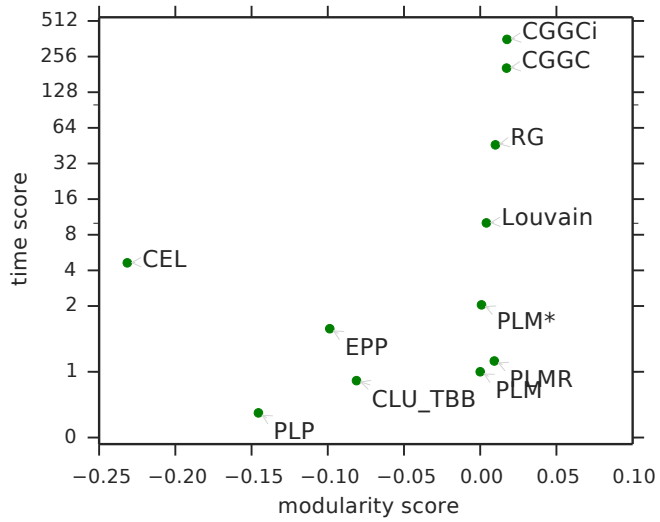


Figure 39: Pareto evaluation of community detection algorithms

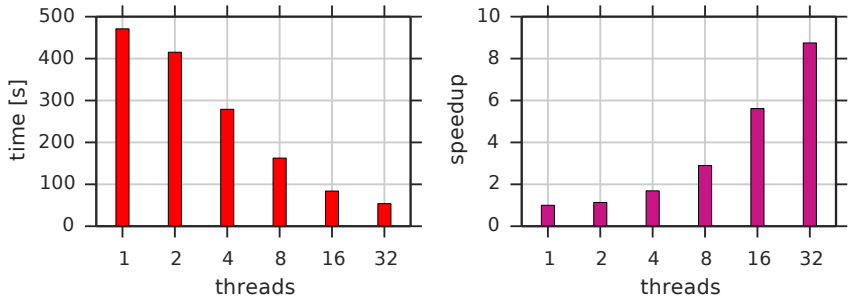


Figure 38: PLP strong scaling on the uk-2007-05 web graph. Left: Absolute running time. Right: Speedup factor over sequential run.

8.4.3 *Parallel Louvain Method (PLM)*

For PLM we observe only small deviations in quality between single-threaded and multi-threaded runs, supporting the argument that the algorithm is able to correct undesirable decisions due to stale data. PLM detects communities with relatively high modularity in the majority of networks. Even large instances are processed in no more than a few minutes. Figure 40 shows the scaling behavior of PLM. Since both the node move phase and the coarsening phase have been parallelized, PLM profits from increased parallelism as well, achieving a speedup of factor 9 for 32 threads. In comparison to PLP (Figure 43b), we observe that PLP can solve instances in only half the time required by PLM, but at a significant loss of modularity. As discussed in Sec. 8.5, the communities detected by the two algorithms can be markedly different. Because the Louvain method for community detection is well-known and accepted, we choose the performance of PLM as our baseline (Figure 43a) and present quality and running time of other algorithms relative to PLM.

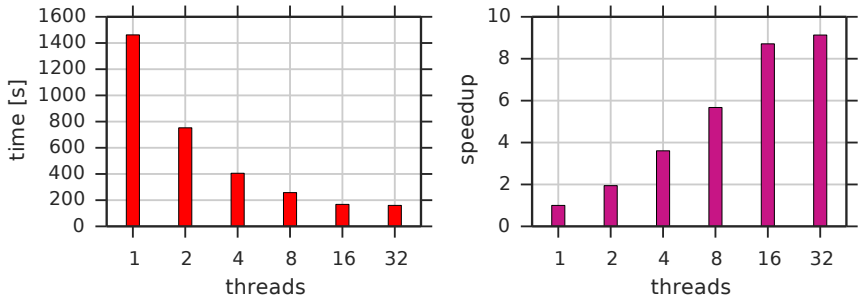


Figure 40: PLM strong scaling on the uk-2007-05 web graph

8.4.4 *Parallel Louvain Method with Refinement (PLMR)*

As shown by Figure 43c, adding a refinement phase generally leads to a (sometimes significant) improvement in modularity. This improvement is paid for by a small increase in running time. The results indicate that our proposed extension of the original Louvain method by a refinement phase can efficiently increase solution quality. We also evaluate the scaling behavior of each phase of the PLMR algorithm. In Figure 41 a yellow bar

indicates the running time on the finest graph while the red bar stops at the total running time of the phase. Time spent on the finest graph clearly dominates all running times. Our experiments show that the move and refinement phases scale well with the number of threads, while the coarsening phase only partially profits from parallelization. The results on this graph are representative for the trend of the scaling behavior for the algorithm's phases: Figure 42 shows speedup factors for each of the phases, aggregated over the test set of 20 graphs.

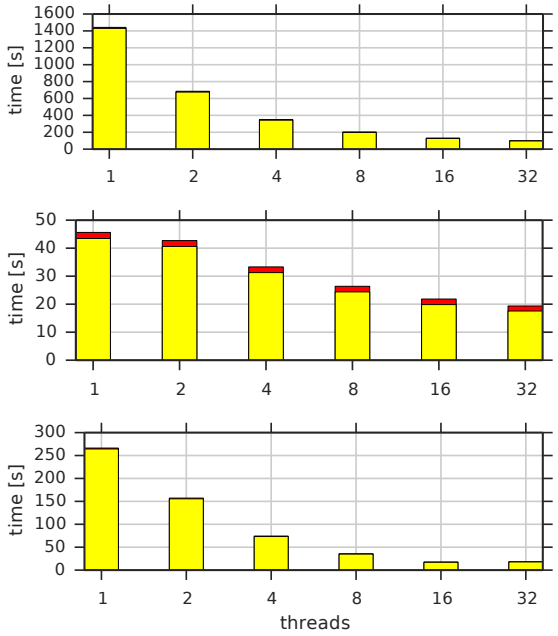


Figure 41: PLMR strong scaling of the move, coarsening and refinement phases (top to bottom) on `uk-2007-05`

8.4.5 *Ensemble Preprocessing (EPP)*

Figure 37 demonstrates the effectiveness of the ensemble approach. Results were generated by an EPP instance with a 4-piece PLP ensemble and PLMR as final algorithm in comparison to a single PLP instance. We observe that the approach of EPP pays off in the form of improved modularity on most instances, exploiting differences in the base solutions and spending extra time on classifying contested nodes. For larger networks, this comes at a cost of about 5 times the running time of PLP alone. It also becomes clear that for small networks the approach does not pay off as running time becomes dominated by the overhead of the ensemble scheme. In comparison to PLM (Figure 43d), the ensemble approach can be slightly faster on some networks, but quality is slightly worse in most cases. We conclude that the ensemble technique EPP is effective in improving on the quality of a single algorithm. While somewhat lower in modularity, the communities detected are similar (see Sec. 8.5) to those of the Louvain method. In practice, our acceleration of the PLM algorithm have made the ensemble approach less relevant.

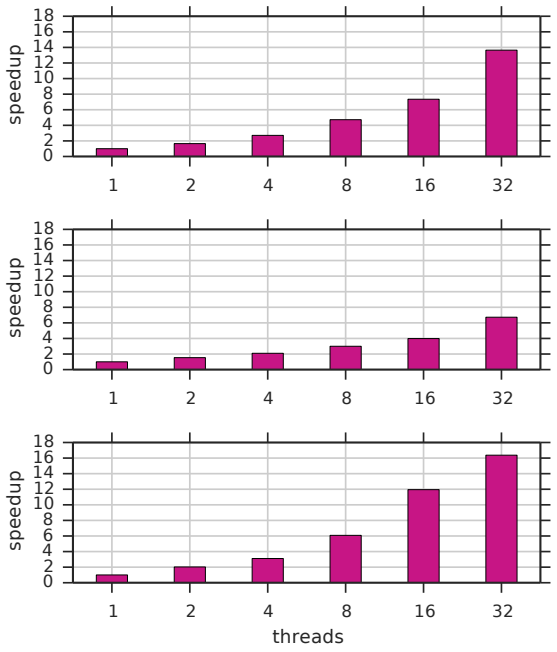


Figure 42: PLMR strong scaling of the move, coarsening and refinement phases (top to bottom) – speedup factors aggregated

8.4.6 *Comparison with State-of-the-Art Competitors*

In this section we present results for an experimental comparison with several relevant competing community detection codes. These are mainly those which excelled in the *DIMACS* challenge either by solution quality or time to solution: The agglomerative algorithms CLU_TBB[4] and RG, as well as CGGC and CGGCi[5], ensemble algorithms based on RG. We also include the widely used original sequential Louvain[6] implementation, as well as the agglomerative algorithm CEL. In contrast to the *DIMACS* challenge, we run all codes on the same multicore machine (Tab. 6) and measure time to solution for sequential and parallel ones alike.

LOUVAIN Although not submitted to the *DIMACS* competition, the original sequential implementation of the Louvain method is still relatively fast (Figure 44a). The marginally different modularity values in comparison to PLM may be caused by subtle differences in the implementation. For example, Louvain explicitly randomizes the order in which nodes are visited, while we rely on implicit randomization through parallelism. For the smallest graphs, running time values are missing because the implementation reported a running time of zero. Louvain eventually falls behind the parallel algorithm for large graphs, confirming that the overhead and complexity introduced by parallelism is eventually justified when we target massive datasets.

4 CLU_TBB http://www.staff.science.uu.nl/faggi101/
5 RG etc: http://www.umiacs.umd.edu/mov/
6 Louvain https://sites.google.com/site/findcommunities/

CLU_TBB AND CEL CLU_TBB, one of the few parallel entries in the *DIMACS* competition, is a very fast implementation of agglomerative modularity maximization, solving the larger instances more quickly than PLM (Figure 44b). Qualitatively however, PLM is clearly superior on most networks. Both in terms of modularity and running time, CLU_TBB occupies a middle ground between PLP and PLM, and is qualitatively very similar to our ensemble algorithm EPP. CEL, as another fast parallel program, produced consistently and significantly worse modularity than PLM, failed to produce a solution on some graphs, and is not as fast as PLP.

RG, CGGC AND CGGCI Ovelgönne and Geyer-Schulz entered the DIMACS challenge with an ensemble approach conceptually similar to what we have developed. Their base algorithm is the sequential agglomerative RG, and two ensemble variants exist: CGGC implements anensemble technique very similar to EPP, while CGGCi iterates the approach. The RG algorithm achieves a high solution quality, surpassing PLM by a small margin on most networks (Figure 44c). Quality is again slightly improved by the ensemble approach CGGC and its iterated version CGGCi (Figure 44d, and 35a), with the latter surpassing any other heuristic known to us. However, all three are very expensive in terms of computation time, often taking orders of magnitude longer than PLM. We consider running times of several hours for many of our networks no longer viable for the scenario we target, namely interactive data analysis on a parallel workstation.

8.4.7 *LFR Benchmark*

As discussed in Sec. 7.3, the LFR benchmark [LFR08] is an established method for evaluating community detection algorithms: A generator produces graphs that resemble real complex networks and contain dense communities which are the more sparsely connected the lower the mixing parameter μ. Algorithm performance is measured as the accuracy in recognizing the ground truth communities supplied by the generator, in view of increasing difficulty (μ). In Figure 45 we plot the agreement (graph-structural Rand index, where 1 is complete agreement) between detected and ground truth communities for our algorithms, and show that the PLM method is able to detect the ground truth even with strong noise ($\mu = 0.8$), while PLP (and hence EPP) is somewhat less robust.

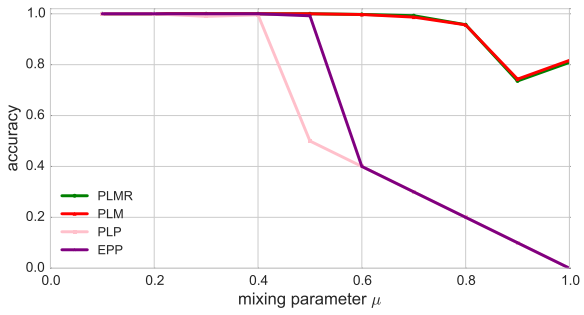
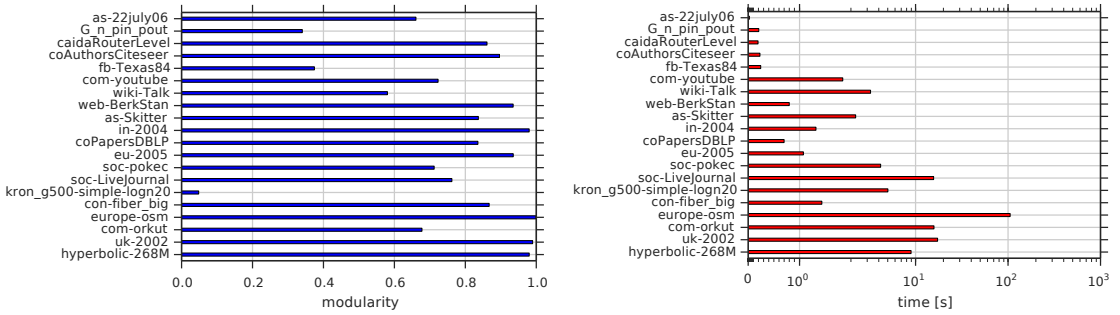
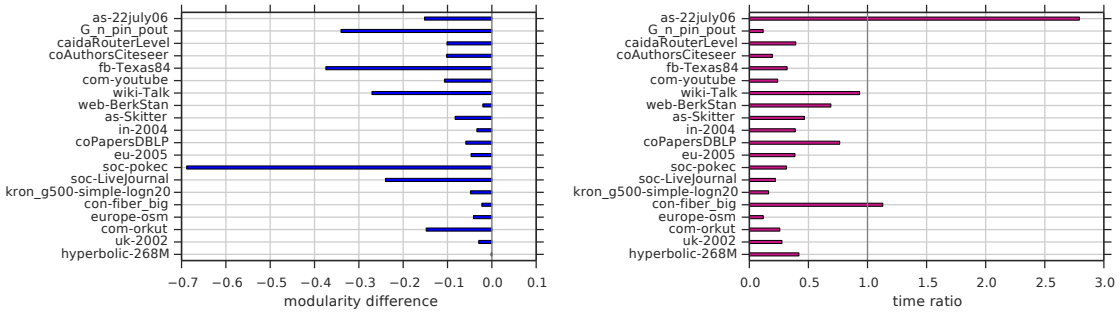


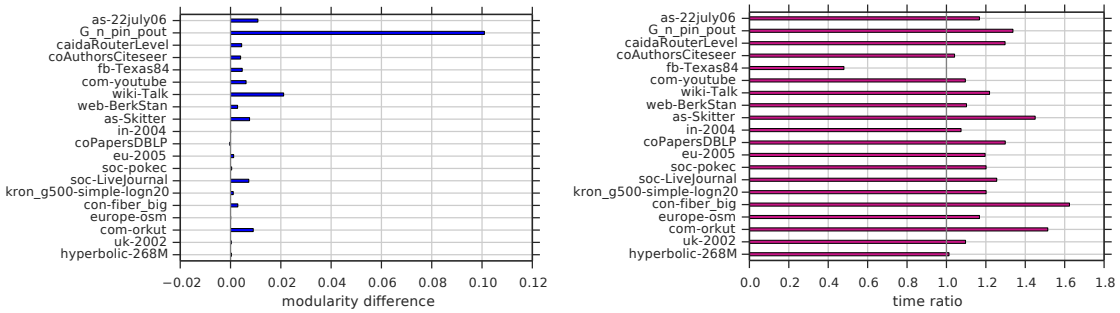
Figure 45: LFR benchmark ($n = 10^5$): accuracy in recognizing ground truth while increasing the number of inter-community edges



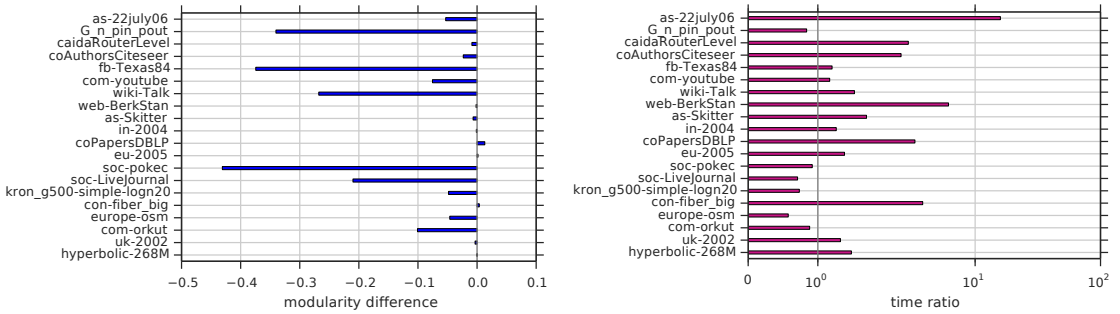
(a) PLM : absolute quality and speed serve as baseline for comparison



(b) PLP

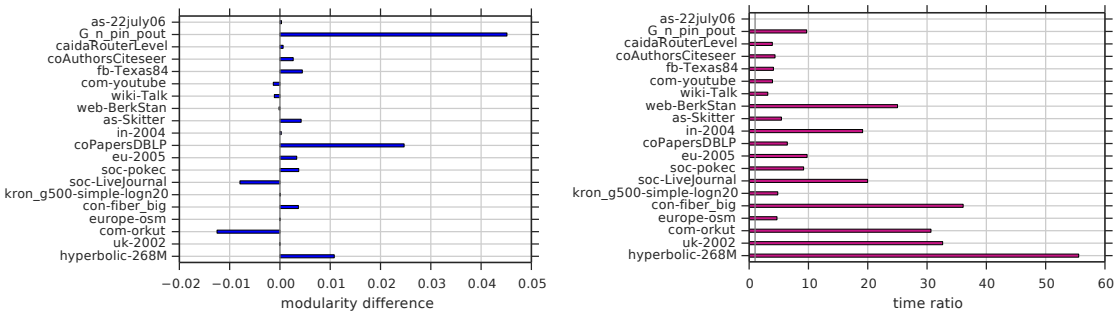


(c) PLMR

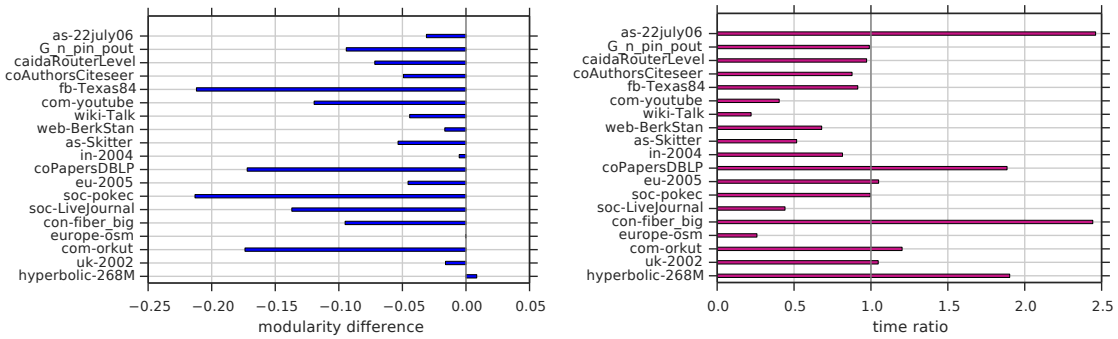


(d) EPP(4, PLP, PLMR)

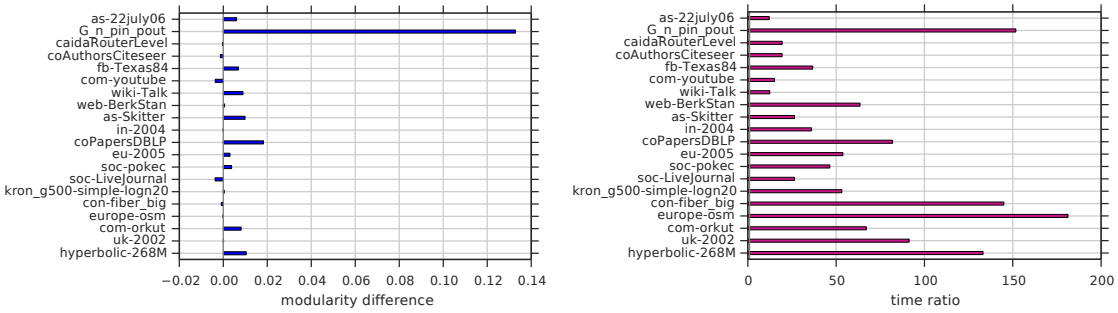
Figure 43: Performance of our algorithms in comparison: PLM serves as the baseline. 32 threads used.



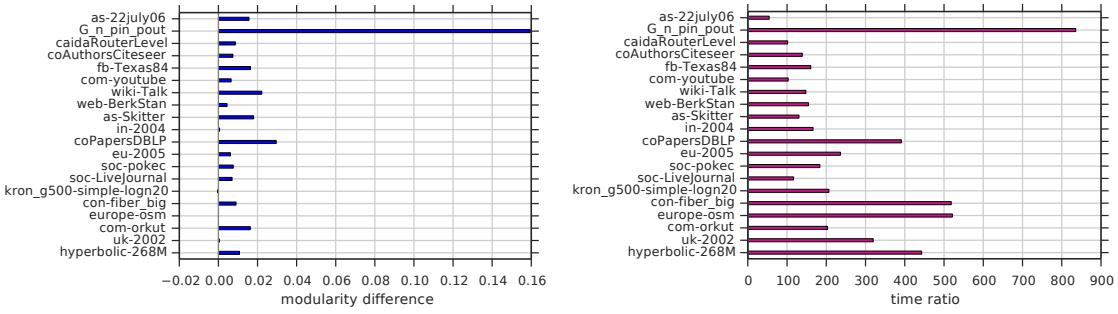
(a) Louvain



(b) CLU_TBB



(c) RG



(d) CGGC

Figure 44: Performance of competitors relative to baseline PLM. 32 threads used for CLU_TBB.

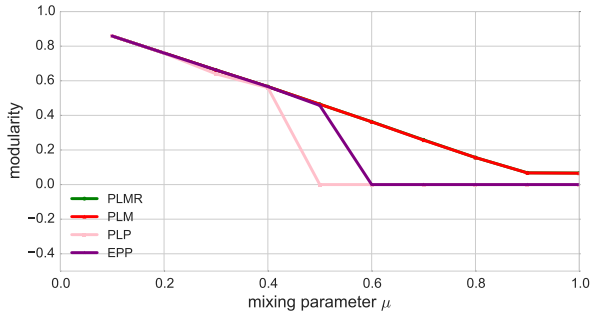


Figure 46: LFR benchmark ($n = 10^5$): modularity while increasing the number of inter-community edges

8.4.8 *One More Massive Network*

In addition to the experiments that went into the Pareto evaluation, we run our parallel algorithms on the web graph `uk-2007-05`, at about 3.3 billion edges the largest real-world data set currently available to us. CLU_TBB fails at reading the input file. This leaves us with five of our own parallel algorithms for Figure 47: EPP(4,PLP,PLMR) takes about 219 seconds, while PLM requires about 156 seconds to arrive at a slightly higher modularity. As expected, PLP is by far the fastest algorithm and terminates in less than a minute. If a certain modularity loss (here 0.02) is acceptable, PLP is also an appropriate choice for quickly detecting communities in billion-edge networks. The processing rate for PLP is over 53M edges/second and over 21M edges/second for PLM with respect to a complete run of each algorithm. These rates confirm the suitability of our algorithms for analyzing massive complex networks on a commodity shared-memory server.

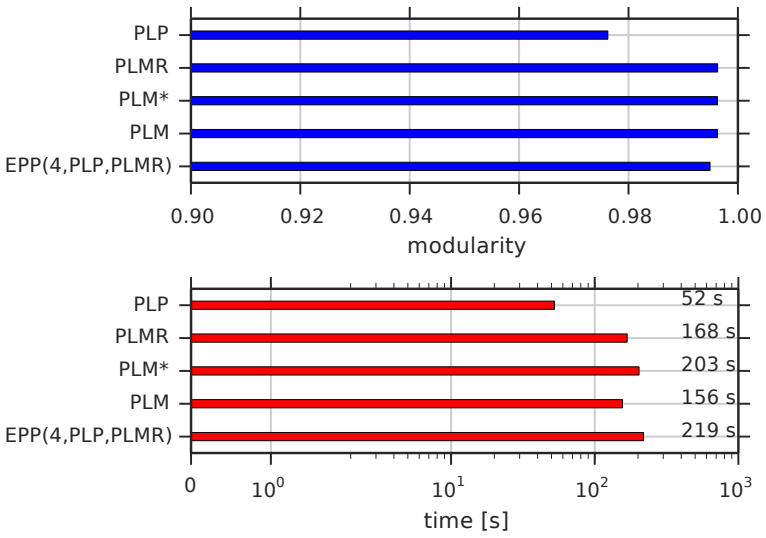


Figure 47: Modularity and running time at 32 threads for our parallel algorithms on the massive web graph `uk-2007-05`

8.4.9 *Weak Scaling*

For weak scaling experiments, we use a series of synthetic graphs where each graph has twice the size of its predecessor (from $\log m = 25 \dots 30$), and double the number of threads simultaneously from 1 to 32. The graphs were created using a generator [vLMP15] based on a unit-disk graph model in hyperbolic geometry [KPK+10] (HUD), which produces both a power law degree distribution and distinctive dense communities. Figure 48 shows the results of weak scaling experiments for PLP and PLM. It must be noted that perfect scaling cannot be expected due to the complex structure of the input. The results of the respective last column have been obtained with hyperthreading, which explains the steeper increase. Figure 49 shows results for additional weak scaling experiments on synthetic graphs generated with the R-MAT model.

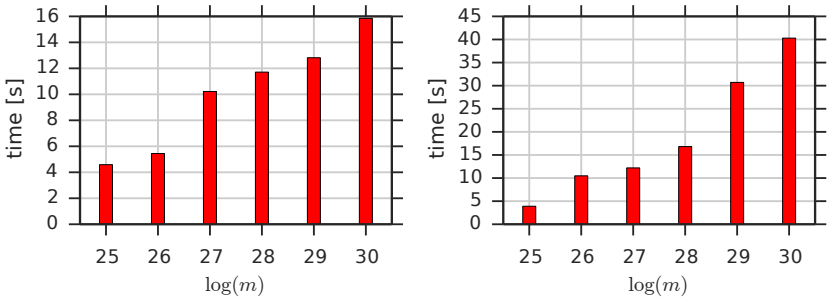


Figure 48: PLP (left) and PLM (right) weak scaling on the series of HUD graphs

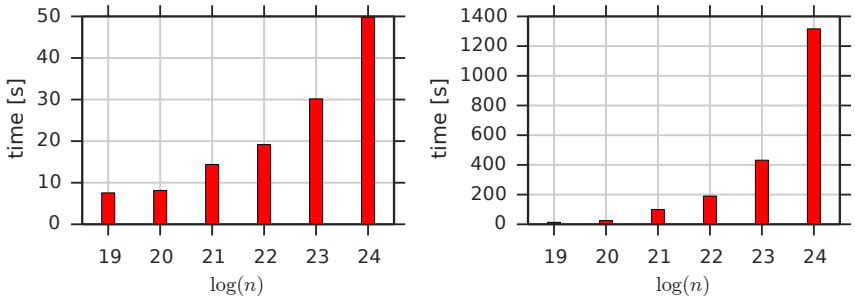


Figure 49: PLP (left) and PLM (right) weak scaling on the series of R-MAT graphs.

8.5 QUALITATIVE ASPECTS

In this work we concentrate on achieving a good tradeoff between high modularity, a widely accepted quality measure for community detection, and low running time. Ideally one should also look for further validation of the detected communities beyond good modularity. As discussed in Sec. 7.3, we do not have a reliable ground-truth partition for real networks, and domain-specific validation of the result goes beyond the scope of this work as we focus on parallelization aspects. Also, most sequential counterparts of our algorithms have been validated before, see [BGLL08]. However, we give an example to illustrate differences between our algorithms in a more qualitative way. Coarsening the input graph according to the detected communities yields a *community graph*, which we then visualize by drawing the size of nodes proportional to the size of the respective

community. Figure 50 shows community graphs for the PGPgiantcompo graph, a social network and web of trust resulting from signatures on PGP keys. The solutions were produced by PLP, PLM, PLMR and EPP(4, PLP, PLMR). It is apparent that has a much finer resolution and detects ca. 1000 small communities. This is true for most of our data sets, but the inverse case also appears. On this network, higher modularity is associated with coarser resolution. , and have a very similar resolution and divide the network into ca. 100 communities. While PGPgiantcompo is admittedly a very small graph, this example shows how community detection can help to reduce the complexity of networks for visual representation.

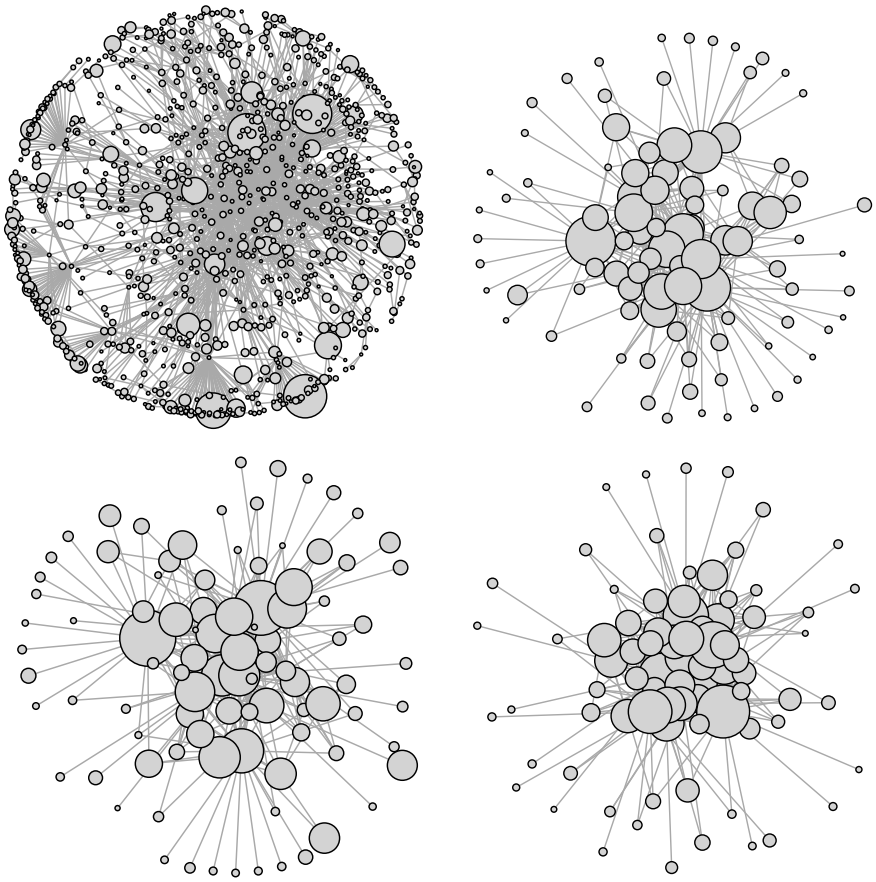


Figure 50: Community graphs of the PGPgiantcompo web of trust for (top to bottom) PLP, PLM, PLMR and EPP(4, PLP, PLMR)

8.6 CONCLUSION

We have developed and implemented several parallel algorithms for community detection, a common and challenging task in network analysis. Successful techniques and parameter settings have been identified in extensive experiments on synthetic and real-world networks. They include three standalone parallel algorithms, all of which are placed on the Pareto frontier with respect to running time and modularity in an experimental comparison with other state-of-the-art implementations. While the label propagation algorithm is extremely fast, its solution might not always be satisfactory for some applications. It

is the first parallel variant of the established Louvain algorithm which can handle massive inputs. On our machine, it detects high-quality communities in a network with 3.3 billion edges in under 3 minutes using 32 threads. Achieving significant parallel speedups over the frequently used sequential algorithm, it can accelerate analysis workflows now and even further on future multicore systems. Our modification of this method adds a refinement phase which enhances modularity for a small increase in running time.

9

DETECTING COMMUNITIES SELECTIVELY AROUND SEED NODES

> A programmer from a very large computer company went to a software conference and then returned to report to his manager, saying: "What sort of programmers work for other companies? They behaved badly and were unconcerned with appearances. There hair was long and unkempt and their clothes were wrinkled and old. They crashed our hospitality suite and they made rude noises during my presentation."
>
> The manager said: "I should have never sent you to the conference. Those programmers live beyond the physical world. They consider life absurd, an accidental coincidence. They come and go without knowing limitations. Without a care, they live only for their programs. Why should they bother with social conventions? They are alive within the Tao."
>
> – from *The Tao of Programming*

The task of selective community detection is concerned with finding high-quality communities locally around seed nodes. Given the lack of conclusive experimental studies, we perform a systematic comparison of different previously published as well as novel methods. In particular we evaluate their performance on large complex networks, such as social networks. Algorithms are compared with respect to accuracy in detecting ground truth communities, community quality measures, size of communities and running time. We implement a generic greedy algorithm which subsumes several previous efforts in the field. Experimental evaluation of multiple objective functions and optimizations shows that the frequently proposed greedy approach is not adequate for large datasets. As a more scalable alternative, we propose selSCAN, our adaptation of a global, density-based community detection algorithm. In a novel combination with algebraic distances on graphs, query times can be strongly reduced through preprocessing. However, selSCAN is very sensitive to the choice of numeric parameters, limiting its practicality. The random-walk-based PageRankNibble emerges from the comparison as a successful and robust candidate.

This chapter is based on joint work with Yassine Marrakchi and Henning Meyerhenke. Results were previously presented at the *First International Workshop on High Performance Big Graph Data Management, Analysis, and Mining* and published as *Detecting Communities around Seed Nodes in Complex Networks* in the proceedings of the *IEEE International Conference on Big Data (IEEE BigData 2014)*.

9.1 INTRODUCTION

In this chapter we consider the problem of quickly finding high-quality communities around given seed nodes, particularly in large complex networks. We refer to this task as *selective community detection* (SCD, also called *local community detection* or *seed set expansion*) to distinguish it from global community detection. In the global scenario, the graph is considered as a whole and an assignment of each node to a community is sought. In contrast, selective community detection begins with a small set of seed nodes as input and finds appropriate communities containing them, obtained by searching the network locally around the seed nodes. Such a targeted approach provides potential for speedup

when solving the problem globally is not necessary or infeasible. Some SCD algorithms also enable us to find query-specific communities, assuming that the best community for seed node s_1 is different from the one for a nearby seed node s_2. SCD methods are especially applicable when we lack global knowledge of the network, e.g. in scenarios where the network structure is discovered on-the-fly. While it may seem that the difference between the global and selective scenario lies only in the amount of work performed, we show that the methods needed are somewhat different. Given these differences, applications are as numerous as for global community detection, and include e.g. finding functional complexes for a given protein in protein interaction networks [VTX09].

The existing literature defines the task in slightly different ways, leading to a diverse set of SCD algorithms. We therefore begin by specifying the task as follows: Given a graph $G = (V, E)$ and a set of seed nodes $S \subset V$, return an assignment of each seed $s \in S$ to a community $C \subset V$ so that s is contained in C and all communities are pairwise disjoint. Within this definition, the following aspects need clarification: We discuss in Sec. 9.3 when a subset of nodes is considered a good community. Depending on the algorithm, the communities may overlap or coincide. Multiple seed nodes can belong to the same community, but every community must contain at least one seed node. We do not require the seed nodes to be structurally close to each other – the considered algorithms can handle both related and unrelated seeds appropriately. Figure 51 illustrates this with an example.

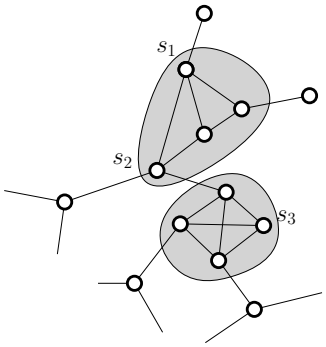


Figure 51: An example of the selective community detection problem and a solution: Seed nodes s_1, s_2 and s_3 were provided as input and the algorithm has detected two communities containing them.

While a variety of methods have been proposed, we observe a lack of conclusive experimental studies demonstrating their practical relevance. With our comparative study, we aim to close this gap. We are also the first to target large complex networks in the order of 10^5 to 10^6 of edges, requiring scalable algorithms and implementations. After a review of previous work on the subject (Sec. 9.2), algorithmic approaches are compared and classified. We identify one widely used approach which we call Greedy Community Expansion (GCE, Sec. 9.4.1). With our generic implementation of GCE, we evaluate existing objective functions and optimizations. In our experimental comparison we include a reimplementation of the random-walk based PageRank-Nibble [ACL06], representing an important class of approaches to the problem. Furthermore, as our main algorithmic innovation, we adapt a global algorithm to the selective scenario (Sec. 9.4.2): A modification of the density-based SCAN algorithm yields our variant selSCAN. The algorithm is generic with respect to a node distance measure, and we propose algebraic distances as

an alternative to the original measure. The performance of algorithms is experimentally evaluated with respect to accuracy, quality, community sizes and running time (Sec. 9.5). Accuracy is measured as the agreement with ground truth communities on synthetic LFR graphs, while quality is calculated with community quality measures which also serve as objective functions for GCE. We then apply the best performing algorithms to large real-world networks.

We deliver an experimental comparison of different classes of SCD algorithms. After reimplementation and extensive analysis, we conclude that a widespread greedy quality optimization method is not likely to perform fast enough on large-scale graphs. A review of previously proposed objective functions narrows the choice for a viable function down to conductance (Φ). We show that the query time can become prohibitively high for networks with millions of edges. Especially for large networks, we propose selSCAN as a faster alternative. In contrast to most other methods, selSCAN also has a notion of outliers. Given an appropriate parameter choice, selSCAN is also qualitatively superior, although efficient parameter selection remains problematic. With selSCAN-AD, we explore a combination of density-based clustering and algebraic distance, which further reduces query time at the cost of precalculating node distances. From the comparison it becomes clear that PageRank-Nibble performs best in terms of running time and result quality.

For SCD we focus only on the communities that contain the seed nodes. In the following, some shorthand terminology is used:

Definition 24 (Core, Boundary and Shell of a Community)**.** Let $E(A, B)$ denote the set of edges $\{u, v\}$ where $u \in A$ and $v \in B$. We denote the set of internal edges of a community as $E_{\text{int}}(C) := E(C, C)$ and its external edges as $E_{\text{ext}}(C) := E(C, V \setminus C)$. Each community C induces three sets of nodes, a core, a boundary, and a shell. The *core* K of C are the nodes in C for which all neighbors are also in C: $K(C) := \{u \in C : v \in C \; \forall \{u, v\} \in E\}$. *Boundary* nodes have neighbors both inside and outside of the community: $B(C) := \{u \in C : \exists \{u, v\} \in E : v \notin C\}$. C is surrounded by a *shell* of nodes which do not belong to C but have edges to nodes in C: $\Omega(C) := \{u \notin C : \exists \{u, v\} \in E : v \in C\}$. □

9.2 LITERATURE OVERVIEW

Though not nearly as extensively studied as the global scenario, SCD has been the focus of multiple previous publications. Because experimental data demonstrating their relative performance is scarce, we aim to clarify the state of the art and summarize and categorize previous efforts.

Community Expansion: A common approach to SCD starts from a seed node as a singleton community and expands the community one node at a time, selecting the candidate which gives the maximum gain for a community quality function. We will refer to this method as greedy community expansion. Examples of this approach are frequent in the literature and vary in the objective function and possible pre- and postprocessing optimizations. Clauset [Cla05], Luo et al. [LWP08], Chen et al. [CZG09] and Bagrow [Bag08] introduce different objective functions as "local modularity" (discussed in Sec. 9.3). An efficient implementation of the first three algorithms runs in $O(|C| \cdot d \cdot |\Omega(C)|)$

where d the average degree of nodes in the community and the last runs in $O(|C| \cdot log|C|)$. Given a set of seed nodes, each seed node is treated separately.

Algorithms Based on Node Similarity: Rather than optimizing community quality one node at a time, this class of algorithms determines the similarity of candidate nodes with a given seed and finds a community among the most similar nodes. A common way to define this similarity is to perform a random walk from a seed, which is likely to get trapped in dense, community-like subgraphs. Nodes are then ordered by the resulting probability distribution and a high-quality community is found among the highest ranked nodes. Instances of this approach include [ST08, ACL06, VTX09]. For our experimental comparison, we implemented the PageRank-Nibble algorithm [ACL06]. Treating the community around a seed node as a graph cut, running time depends on the size of the small side of the cut rather than the size of the input graph. The algorithm offers theoretical guarantees with respect to the conductance of the cut.

Other Approaches: Apart from these main classes of algorithms, other approaches have also been explored. An example is bridge-bounding [PSV+09], which starts with a singleton community around a seed and attaches nodes from the shell as long as the edges connecting these nodes are not bridges. Bridges are defined by globally scoring edges by centrality (e.g. by betweenness centrality). One of the few works targeting large networks [RBJ+11] proposes an agglomerative algorithm: Starting from a handful of seed nodes and singleton communities, mergers are performed between communities containing a seed node and communities adjacent to them.

9.3 MEASURING COMMUNITY QUALITY

In the following we explain and justify our choice of measures for the quality of a seeded community C_s. The measures we select then serve both as objective functions in GCE and quality criteria for the evaluation. On the right side of each function we denote the value range and whether it is maximized or minimized (e.g. $[0, 1]$ min).

Discarding Modularity: For global community detection, modularity has proven to be effective, despite a few drawbacks (see Section 7.1). We considered modularity in the context of selective community detection, but decided against it: Modularity depends strongly on the global structure of the graph via the number of edges and the global null-model and expects a full graph partition. In the remainder we examine community quality measures for a single community without global knowledge of the network.

Conductance: Consider a single community as a set of nodes cut from the rest of the graph. Sparsity of the cut is a necessary criterion for a good community. One important cut measure is conductance, the size of a cut divided by volume of the smaller section of the graph.

Definition 25 (Conductance)**.** For a graph $G = (V, E)$, the *conductance* $\Phi(C)$ of a subset of nodes C is defined as

$$\Phi(C) := \frac{|E(C, V \setminus C)|}{\min\{vol(C), vol(V \setminus C)\}} \tag{20}$$

$$\underset{|C| \ll |V|}{=} \frac{|E(C, V \setminus C)|}{vol(C)} \qquad [0, 1] \text{ min} \tag{21}$$

□

Minimizing the cut alone encourages small communities, while maximizing the volume requires larger ones. Minimizing conductance therefore helps to identify non-trivial subsets of nodes which are sparsely connected to the rest of the graph. It should be noted that determining the minimum conductance cut in a graph is $\mathcal{NP}$-hard [ŠS06]. Conductance is regularly used for measuring community quality locally (e.g. [VTX09, AL06]).

Custom Measures for SCD: Considering that modularity is ill-suited as an objective function for SCD, several alternatives have been proposed under the name of "local modularity". In spite of the name, they are not directly related to each other and should not be considered localized variants of global modularity, since they do not rely on a null model. We will refer to these measures by their abbreviations in the literature instead.

Clauset [Cla05] defines R as a ratio of boundary edges to community nodes and all edges connected to boundary nodes:

Definition 26 (R-measure)**.** Given a graph $G = (V, E)$ and a community $C \subset V$, the R-measure is defined as

$$R(C) := \frac{|E(B(C), C)|}{|E(B(C), V)|} \qquad [0, 1] \text{ max} \tag{22}$$

□

Luo et al. [LWP08] propose M, which is the ratio of intra-community edges to inter-community edges:

Definition 27 (M-measure)**.** Given a graph $G = (V, E)$ and a community $C \subset V$, the M-measure is defined as

$$M(C) := \frac{|E_{int}(C)|}{|E_{ext}(C)|} \qquad [0, \infty] \text{ max} \tag{23}$$

□

For the average internal degree in the community and the average external degree in its boundary, we obviously want to maximize the former and minimize latter, resulting in the L measure introduced by Chen et al. [CZG09].

Definition 28 (L-measure)**.** Given a graph $G = (V, E)$ and a community $C \subset V$, the L-measure is defined as

$$L(C) := \frac{2 \cdot |E_{int}(C)|}{|C|} \cdot \left(\frac{|E(B(C), \Omega(C))|}{|B(C)|} \right)^{-1} \qquad [0, \infty] \text{ max} \tag{24}$$

□

Selection of Measures: Previous empirical results [Bra10, WHHF12] show that M yields consistently better results than optimizing R and the measure proposed by Bagrow in terms of agreement with ground truth data. Furthermore, we show the equivalence of M and Φ as objective functions, a relationship not observed in [LWP08].

Lemma 1. *For a graph without self-loops and $|C| \ll |V|$, the following holds:*

$$\min! \, \Phi(C) \iff \max! \, M(C) \tag{25}$$

i.e. we maximize C exactly if we minimize Φ.

Proof.

$$\begin{aligned}
\min!\,\Phi(C) &\iff \max!\,\Phi(C)^{-1} \\
&\iff \max!\,\left(\frac{|E_{\text{ext}}(C)|}{2\cdot|E_{\text{int}}(C)|+|E_{\text{ext}}(C)|}\right)^{-1} \\
&\iff \max!\,\left(1+2\cdot\frac{|E_{\text{int}}(C)|}{|E_{\text{ext}}(C)|}\right) \\
&\iff \max!\,M(C)
\end{aligned}$$

□

Therefore, we will evaluate Φ and L as community quality measures and use M and L as objective functions for GCE to maximize.

9.4 ALGORITHMS

9.4.1 *GCE: Greedy Community Expansion*

In the existing literature on SCD, many of the proposed algorithms are variations of one basic approach which we call greedy community expansion: A community C_s around seed s is expanded by including the shell node which yields the highest gain ΔQ with respect to some community quality measure Q. After each inclusion, gains are recomputed for all candidates. Previously proposed greedy algorithms (e.g. [Cla05, LWP08, CZG09, WHHF12, Bag08]) vary in the objective function as well as different pre- and post-processing steps and optimizations. Many of these variants are similar enough to be implemented as one generic algorithm with exchangeable components. This yields GCE, which we use to evaluate the greedy method with respect to large complex networks. GCE has an exchangeable objective function as well as an optional optimization which we call acceptability. It is claimed to improve quality by prioritizing certain candidate nodes [WHHF12]. As explained in Sec. 9.3, we implement both L and M (equivalent to Φ) as objective functions. We indicate the configuration with the following naming scheme: For example, GCE-L optimizes L. In the following paragraphs, we give more details on objective functions and optimizations.

Objective Functions: We can efficiently calculate the quality gain for candidate nodes $v \in \Omega(C)$ in $O(\deg(v))$ by keeping track of interim values like the number of internal and external edges and the number of boundary nodes. Let $Q'(C)$ be the current community quality and let $\deg_{\text{int}}(v,C) := |\{u \in C \cup \{v\} : \{u,v\} \in E\}|$ and $\deg_{\text{ext}}(v,C) := \deg(v) - \deg_{\text{int}}(v,C)$.

Then the gain for M and L when moving the shell node v to the community C is

$$\Delta M(v,C) = \frac{|E_{\text{int}}(C)| + \deg_{\text{int}}(v,C)}{|E_{\text{ext}}(C)| - \deg_{\text{int}}(v,C) + \deg_{\text{ext}}(v,C)} - M'(C) \tag{26}$$

$$\Delta L(v,C) = \frac{\Delta L_1(v,C)}{\Delta L_2(v,C)} - L'(C) \tag{27}$$

where

$$\Delta L_1(v, C) = \frac{2 \cdot (|E_{\text{int}}(C)| + \deg_{\text{int}}(v, C))}{|C| + 1} \tag{28}$$

$$\Delta L_2(v, C) = \frac{|E_{\text{ext}}(C)| - \deg_{\text{int}}(v, C) + \deg_{\text{ext}}(v, C)}{|B(C)| + \Delta|B(C)|} \tag{29}$$

Each expansion step is followed by recalculation of the gains in $O(|\Omega(C)| \cdot d)$ where d is the average degree of shell nodes.

9.4.2 *selSCAN: a Density-based Approach*

As an alternative to community expansion we propose our adaptation of SCAN (Structural Clustering Algorithm for Networks) [XLJ+12], a global community detection algorithm inspired by the data-clustering algorithm DBSCAN [EKSX96]. SCAN tries to transfer the characteristics of DBSCAN to community detection in networks. We briefly revisit the concepts on which SCAN is based. It operates with a similarity measure for pairs of connected nodes—we define it equivalently in terms of node distances in order to combine it with algebraic distances. The ϵ-neighborhood $N_\epsilon(v) = \{u \in N(v) : d(u, v) < \epsilon\}$ is the set of close neighbors which have at most ϵ distance from v. A node is a core if it has more than κ close neighbors, formally $\text{core}_{\kappa,\epsilon}(v) \iff |N_\epsilon(v)| \geq \kappa$. Two nodes from the graph are called density-connected if they are joined by a path where at least all inner nodes are cores: A SCAN community is then a maximal set of density-connected cores and their close neighbors. All remaining nodes are either hubs or outliers, i.e. central nodes lying between two or more communities or peripheral nodes not belonging to any community. Figure 52 illustrates this with a small example.

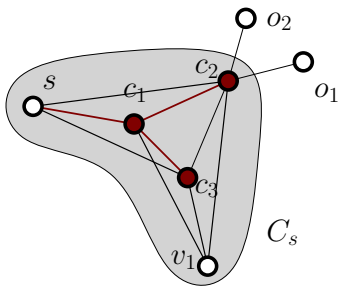


Figure 52: Example of a SCAN-community, spanned spanned by the core nodes c_1, c_2 and c_3, including their close neighbors v_1 and the seed s. Nodes o_1 and o_2 are outliers.

SCAN runs in $O(m)$ time. We continue by describing selSCAN, our adaptation of SCAN to the SCD scenario. selSCAN receives as input a set of related or unrelated seed nodes and returns an assignment of seed to community, but in contrast to GCE each pair of communities is either disjoint or identical. Algorithm 6 denotes selSCAN in pseudocode. The algorithm considers each seed in turn, but can recognize that a seed belongs to a previously discovered community, saving time for sets of related seeds (line 1). Our adaptation differs from the original SCAN in the following aspects: We begin with seed nodes rather than random nodes. If a seed s is not a core, selSCAN tries to find a core in its neighborhood (line 8). If one is found, a community is constructed around it as denoted in Algorithm 7, which is best understood as a breadth-first search among close neighbors

where only cores are added to the search queue. If no core is found, s is classified as an outlier (line 14). A distinction between hubs and outliers cannot be made, because this would require a global partition.

A downside which SCAN and selSCAN inherit from DBSCAN is that the method is not parameter-free: The result depends on the parameters κ and ϵ, which need to be estimated per network. We are aware of an algorithm [SHH+10] which has a similar community concept as SCAN and automatically determines an ϵ which is favorable for a community quality measure, but is based on a global spanning tree of the graph and deviates too strongly from the SCD scenario.

Algorithm 6: selSCAN: Selective Structural Clustering Algorithm for Networks

```
Input: Graph G = (V, E), node distances d, seed set S, parameters κ, ϵ
Output: η: assignment of seed s ∈ S to community C_i or outlier status ∅, default value
        undefined ⊥
1  for s ∈ S : η(v) =⊥ do
2      create queue Q
3      if core_{κ,ϵ}(s) then
4          η(s) ← new community C
5          enqueue(Q, s)
6          coreSearch(Q,C)
7      else
8          if ∃c ∈ N_ϵ(s) : core_{κ,ϵ}(c) then
9              c ← core with minimum distance d(s, c)
10             η(s) ← new community C
11             enqueue(Q, c)
12             coreSearch(Q,C)
13         else
14             η(s) ← ∅
15 return ζ
```

Algorithm 7: coreSearch(Q,C)

```
1  while Q not empty do
2      x ← dequeue node from Q
3      for y ∈ N_ϵ(x) do
4          if η(y) =⊥ then
5              η(y) ← C
6              if core_{κ,ϵ}(y) then
7                  enqueue(Q, y)
```

Node Distance Measures: selSCAN is generic with respect to a node distance measure as defined in Sec. 2.2. The original SCAN is implemented with a node distance measure which expresses the amount of overlap between node neighborhoods:

$$ND(u,v) = 1 - \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)| \cdot |N(v)|}} \tag{30}$$

In addition to ND, we employ algebraic distance (AD) as concept for expressing the structural closeness of nodes. As described in Sec. 2.2, algebraic distance is a way of measuring the connection strength between a pair of nodes which are not necessarily neighbors [CS11], and tends to be low for pairs of nodes that are located in a common dense subgraph. $ND(u, v)$ can be calculated on the fly when discovering a community, but only takes the direct neighborhoods into account. $AD(u, v)$ requires a preprocessing phase, but can incorporate more structural information. An efficient implementation is described in Sec. 5.1.1. Note that AD preprocessing can also be implemented with a distributed algorithm because it involves only computations between neighboring nodes.

9.5 EVALUATION

We experimentally test the performance of *GCE-M*, *GCE-L*, *selSCAN* and *PageRankNibble* in terms of accuracy, quality and running time to evaluate whether they are appropriate for large complex networks. In extensive experiments, a subset of which is presented here, we used both synthetic LFR graphs and real-world social networks. Implementations are written in C++ and extending *NetworKit*. Experiments are performed on a compute server with 2 x 8 Cores: Intel(R) Xeon(R) E5-2680 0 at 2.70GHz, 32 threads and 256 GB RAM. The compiler is GCC 4.7.1 with -O3 optimization.

9.5.1 *LFR Benchmark*

As described in Sec. 7.3, the LFR graph generator provides reliable ground truth communities to evaluate the performance of community detection methods, also applicable in the context of selective community detection. For the evaluation, we vary the LFR mixing parameter μ, which stands for the fraction of inter-community edges. For higher mixing parameters, communities are less dense and their boundaries are less clearly defined, increasing the difficulty of the task. Distinctive communities can be identified up to $\mu = 0.5$. We quantify the accuracy of an algorithm in recognizing the ground truth in the following way: Let C_s be the detected community for seed s and T_s its ground truth community. We use the Jaccard index $J(C, T) := \frac{|C \cap T|}{|C \cup T|}$ to quantify their agreement. Jaccard values closer to 1 are better, and containment of T within C is not enough, because the Jaccard index penalizes the size of the set $C \setminus T$.

9.5.2 *Parameter Studies*

Both *PageRankNibble* and *selSCAN* depend on numeric parameters, the choice of which is crucial for community detection success. For *PageRankNibble*, we need to set α and β: α is the loop probability of the random walk, and smaller values tend to produce larger communities. β is the tolerance threshold for approximation of PageRank vectors, and smaller β leads to more accurate approximation and higher running time. For *selSCAN*, the parameters to estimate are ϵ, the threshold distance for two neighboring nodes to be considered close, and κ, the number of close neighbors required to be a core node. When using algebraic distances, we need to additionally estimate the number of iterations required for meaningful node distances, because values are increasingly smoothened with each iteration. The distance threshold ϵ needs to be selected depending on the resulting

absolute distance values. In general, parameter dependence is a disadvantage of those methods, since community detection is usually an exploratory data analysis task and reliable ground truth against which to tune the parameters is not available. However, the LFR generator provides reliable ground truth, and we perform the following parameter study: For an LFR graph, we measure which combination of two parameters yields the best agreement with ground truth. By setting parameters this way, the algorithms are configured to recognize community structure that it similar to that found in the LFR benchmark graphs: Community sizes between 50 and 250 nodes are typical for actual social networks. From the results of the parameter study it became clear that we can fix κ at 2 and select an appropriate ϵ. Using the ND distance function, ground truth communities were best recognized for $\epsilon = 0.75$. For algebraic distances, we observed that 10 iterations and a distance threshold of $\epsilon = 0.01$ performed best. For *PageRankNibble* an approximation tolerance of 10^{-4} is sufficient and a loop probability $\alpha = 0.1$ performs well. We apply the same parameters to the real world networks in Section 9.5.5.

9.5.3 *LFR Benchmark Results*

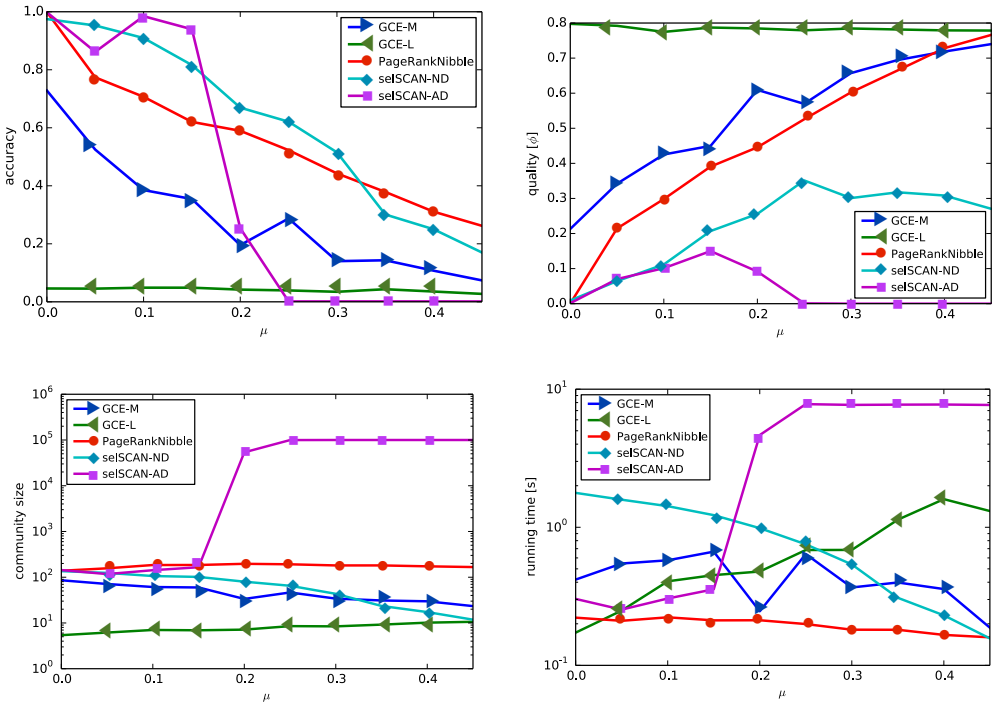


Figure 53: LFR benchmark results: accuracy, quality, community size and running time (query time without preprocessing)

Fig. 53 contains plots of LFR benchmark results. We let each algorithm find communities for 100 seed nodes in a 10^5 node LFR graph and record average accuracy, average community quality in terms of conductance, average community size and query time for the batch of seed nodes. Running times do not include the preprocessing phase of *selSCAN-AD*, which we discuss in more detail in Sec. 9.5.5. Additional experiments show that the results do not qualitatively change when we scale up the number of nodes in the

graphs. The crucial factor is the size of the ground truth communities (here between 50 and 250 nodes) and the ratio of intra- to inter-community density controlled by the parameter μ. We increase the difficulty for algorithms by increasing μ up to 0.5. Algorithms tend to perform worse for increasing μ, the LFR parameter influencing the amount of inter-community edges. We consider those algorithms superior which can distinguish communities even with significant noise. *GCE-L* terminates without discovering the ground truth community, reflected in small community sizes. *GCE* variants optimizing L run slightly faster than those using M. In terms of accuracy, M clearly outperforms L. L leads to a low agreement with LFR ground truth. Inspecting precision and recall shows that on average about 80% of nodes are correctly assigned by *GCE-L*, but C_s is only insufficiently expanded to half the size of T_s. It seems to be a general limitation of L that big communities are not discovered because expansion halts after few iterations. This can be explained by the fact that the denominator L_{ext} grows faster than the numerator L_{int} during expansion. The same occurs for the non-greedy *CE-L* algorithm. In view of these results, we cannot consider the L measure an appropriate objective function for our purposes. *GCE-L* running times increase with μ because they are coupled to the growth of the shell. The same is not true for the density-based *selSCAN-ND* algorithm, whose running time even decreases as the size of discovered communities decreases. Qualitatively, *selSCAN-ND* performs well and behaves robustly with increasing noise between the communities, even outperforming *PageRank-Nibble*. *selSCAN-AD* starts as one of the fastest algorithms on the LFR graphs. It avoids the (re)computation of quality gains and can also return a result in constant time if the seed belongs to a previously discovered *SCAN* community. Query time is slightly faster than for *selSCAN-ND*, because node distances are calculated as pre-processing (see Sec. 9.5.5 for discussion). It is important to note, however, that the high accuracy of *selSCAN* depends on the right choice for the parameter ϵ (see Sec. 9.5.2). Clearly, accuracy of *selSCAN-AD* peaks for the μ value on which the parameters have been trained, and then plummets. At the same time the size of the detected community escalates to encompass most of the network. These results indicate that *selSCAN* performs well only for correctly selected parameters, with a narrow margin of error. *PageRank-Nibble* needs parameter tuning as well, but is more robust and convinces with very low query times and a relatively high accuracy throughout the experiments.

9.5.4 *Real-world Social Networks.*

We compare the algorithms on the "Facebook 100" collection of real social networks collected in the early days of Facebook [TMP12] (cf. Chapter 10 for more details). Table 8 contains graph sizes, an estimate of the average local clustering coefficient c, and the exponent γ of the degree distribution power law.

9.5.5 *Results for Real-World Networks*

For experiments on real-world networks, we select *GCE-M*, *GCE-L*, *selSCAN-ND* , *selSCAN-AD* and *PageRank-Nibble* as candidates. The results shown in Fig. 54 are averages of community quality (Φ), community size and total running time for a seed set of 10 nodes. Running times do not include *AD* preprocessing time. Because of the lack of reliable ground truth data, we cannot test accuracy. Quality as measured by conductance must

no	name	n	m	c	γ
0	Caltech36	769	16656	0.429	4.94
1	Smith60	2970	97133	0.291	10.44
2	Johns Hopkins55	5180	186586	0.276	4.37
3	UChicago30	6591	208103	0.261	3.78
4	Carnegie49	6637	249967	0.283	5.33
5	MIT8	6440	251252	0.279	3.84
6	Princeton12	6596	293320	0.240	4.87
7	Yale4	8578	405450	0.240	4.49
8	Harvard1	15126	824617	0.225	3.13
9	Oklahoma97	17425	892528	0.233	7.40

Table 8: Overview of real-world social networks used. c: average local clustering coefficient, γ: exponent of degree distribution power law

be considered in combination with community sizes. We observed that good values of Φ can also be achieved for communities which encompass large parts of the graph, which are not likely to be desired solutions.

Strikingly, query times for *GCE-M* and *GCE-L* escalate completely on these real networks, being orders of magnitudes larger than the query times of *selSCAN-AD* and *PageRank-Nibble*, which are close to zero on this scale. We conjecture that this is due to high-degree nodes leading to an extreme growth of the size of the shell to be scanned. The resulting query times can only be called impractical. Query time for *selSCAN-ND* is also very high, for similar reasons as the overlap of large neighbourhoods has to be calculated. *selSCAN-AD* has potentially the fastest query times due to the precalculation of node distances, so that actual queries amount to little more than breadth-first search. While the preprocessing time is not insignificant, it can amortize if repeated queries on the same network need to be performed. We observe that AD preprocessing time is linear in the number of edges. In practice, however, both *selSCAN-ND* and *selSCAN-AD* fail with the parameters trained on LFR graphs: *selSCAN-ND* produces giant communities while *selSCAN-AD* recognizes only outliers. This shows how sensitive the density-based approach is to the choice of numeric parameters, especially the distance threshold ϵ. Values that achieved high accuracy on the LFR set fail completely for this set of real world social networks, although we can assume at least some similarity with respect to community structure and graph properties. In contrast, *PageRank-Nibble* performs rather well and in an expected way using the parameters estimated from LFR experiments.

9.6 CONCLUSION

While various methods for selective community detection have been proposed, there is a gap in the literature with respect to an experimental comparison on real-world data. We contribute to the consolidation of the topic by reviewing several algorithms and objective functions proposed for the selective community detection problem. We highlight the need for scalable solutions, showing that a popular greedy approach is not fast enough to target large complex networks in the order of millions to billions of edges, which are not uncommon today. Greedy node-by-node optimization of community quality may not be

the best strategy at all, since the process can easily halt in unwanted local optima. We propose the density-based *selSCAN* as an alternative approach, which can be more accurate and faster especially when combined with algebraic distances. Although algebraic distances require preprocessing, they take more structural information into account and provide us with node distances appropriate for community detection. Calculating algebraic distances in a preprocessing step allows us to keep query times in the order of a few milliseconds even for graphs with millions of edges. The density-based approach is, however, very sensitive to the choice of a numeric parameter ϵ, which depends on the specific properties of a network. Solving the problem of parameter estimation without previous knowledge would allow us to harness the strengths of the density-based approach in practice. Finally our experiments show that the *PageRank-Nibble* algorithm is likely to be a practical method for selective community detection. Although the algorithm depends on numeric parameters that have to be estimated in parameter studies, these seem to be more robust and can be carried over to real-world networks. An efficient implementation of the algorithm is distributed as part of NetworKit.

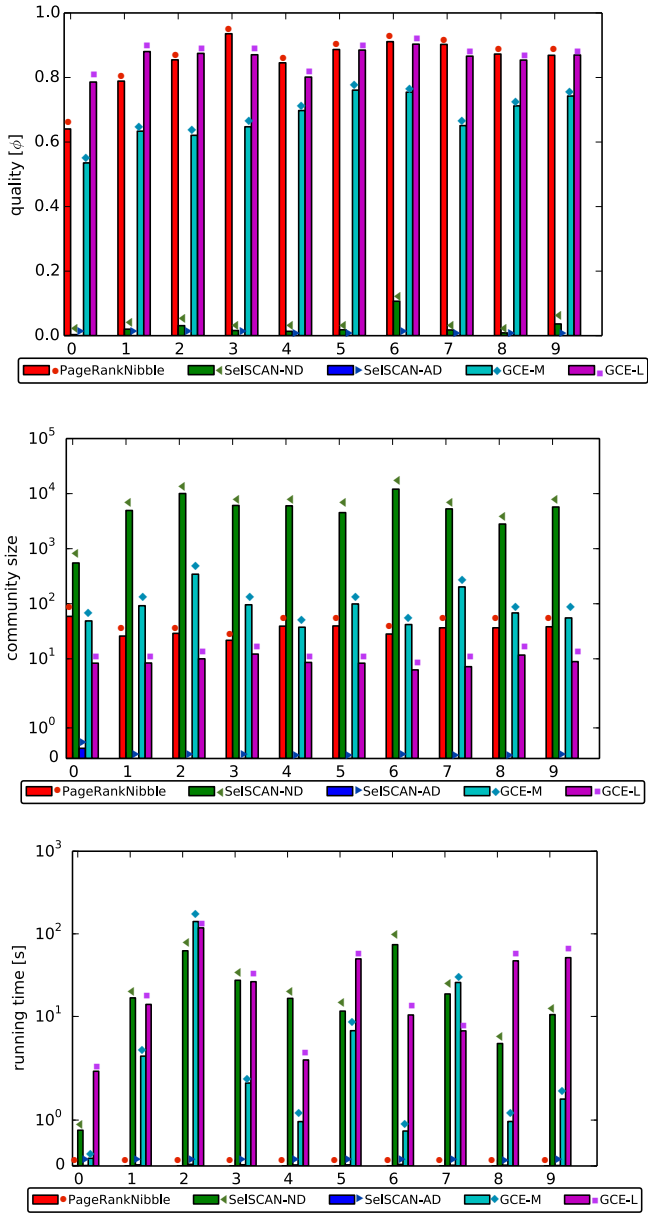


Figure 54: Average community quality, size and running time for real complex networks (Table 8)

CONCLUSION OF PART III

Community detection, the subdivision of a network into subsets of nodes with internally dense and externally sparse linkage, is a fundamental analysis task that reveals the modular composition of a network. In Part III we have approached two subproblems, detecting disjoint communities either globally, or selectively around given seed nodes. Though they may seem initially similar, we have seen that these two problems require different approaches in terms of formalization and algorithmic techniques.

Global disjoint community detection was formalized as the $\mathcal{NP}$-hard modularity optimization problem. Two efficient parallel heuristics (both based on previously published sequential heuristics) have been engineered and experimentally evaluated: The PLM (Parallel Louvain Method) algorithm targets modularity explicitly, combining locally greedy node moves with a multilevel scheme. A variant with additional local moves during the refinement yields a small modularity gain. The PLP (Parallel Label Propagation) algorithm is a simple, fast and easily parallelizable method that implicitly leads to high-modularity.

The experimental comparison on a wide range of complex networks locates both algorithms on the Pareto front of modularity versus running time in comparison with several competing implementations, with PLM algorithm leading to higher modularity than PLP at a moderate additional cost in running time. In summary the work presented resulted in two robust, $O(m)$-time algorithms for modularity-driven community detection in large networks on a shared-memory parallel computer, which show efficient scaling behavior and are parameter-free by default. Assuming as hardware a current multicore workstation, users of NetworKit can deploy as community detection tools for large-scale networks consisting of several billion edges, and process them in no more than a few minutes. Their workflows are therefore constrained by the main memory available to store the graph rather than the computing time that needs to be invested, and the graph representation in memory is already compact. Consequently, from a pragmatic perspective that is concerned with building a practical tool set for network analysis on a multicore machine, the developed methods present a satisfactory solution. Further speedups within this context will likely yield diminishing returns in terms of the expansion of the user's data analysis capabilities. To process significantly larger graphs, we need to look for external memory algorithms, distributed computing solutions or graph compression schemes, which is outside of the scope of the NetworKit framework as well as this thesis.

From this pragmatic perspective, the selective community detection problem cannot yet be considered solved to the same extent. Since previous literature is rich in proposals of novel algorithms and objective functions, but poor in terms of rigorous evaluation and comparison, the state of the art was initially less clear. A significant part of the work was concerned with clarifying the state of the art and developing standards for comparison. Engineering efforts yielded the algorithms GCE (Greedy Community Expansion) which unifies several previously proposed algorithms and is able to target different objective functions, and selSCAN, an adaptation of a global, density-based community detection scheme to the the selective community detection scenario. For practical purposes, we would like such an algorithm to detect communities selectively in a small fraction of the time needed for solving of the global solution. Since global methods (including those

introduced in Chapter 8) are already very fast, there is a need for performance improvement in the case of the selective methods. We also require an SCD algorithm not to be dependent on complex parametrization, so that it is suitable for exploratory analysis. In contrast, the PageRank-Nibble and selSCAN algorithms are parameter-sensitive and required parameter studies to perform well.

Beyond the algorithmic horse race enabled by narrowing the focus to optimizing a target function like modularity, the main open question that remains is to better specify when a result is adequate. If two community detection methods differ both in running time and qualitatively in their results – which one should be selected? Application-specific feedback could perhaps lead to a better understanding of the tradeoff between running time and result quality. The question of an optimal tradeoff is one that the algorithm specialist cannot leave entirely to the practitioner: While the tradeoff may be to some extent user-configurable (such as the optional refinement phase of PLMR), it is usually at least partially fixed by certain algorithm design choices.

Part IV

EDGE CENTRALITY MEASURES FOR NETWORK SPARSIFICATION

10 RATING THE CENTRALITY OF EDGES

> One day a student came to Moon and said: "I understand how to make a better garbage collector. We must keep a reference count of the pointers to each cons."
>
> Moon patiently told the student the following story:
>
> "One day a student came to Moon and said: 'I understand how to make a better garbage collector...
>
> – traditional hacker koan

This chapter examines the idea that different connections play different roles with respect to structural properties of the network, and discusses ways to quantify their structural importance. Several existing (Triangle Count, Jaccard Similarity, Simmelian Backbones, Algebraic Distance) and original (Local Degree, Edge Forest Fire) methods are described, including their efficient implementation. An experimental study is presented which focuses on forming classes of methods according to how the resulting edge ranks correlate in a large set of social networks. In the following Chapter 11, these methods are applied and evaluated for the goal of network sparsification, also a context in which many of these measures have been first proposed. We conceptualize sparsification as edge rating followed by filtering. Sparsification should be considered one possible application of rating the structural importance of edges.

This chapter and the subsequent chapter are based on joint work with Gerd Lindner, Michael Hamann, Henning Meyerhenke and Dorothea Wagner. Gerd Lindner did important preliminary work in the course of his Bachelor's thesis on *Complex Network Backbones* [Lin14]. Results appeared as *Structure-Preserving Sparsification of Social Networks* in the proceedings of the *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2015)* [LSH+15]. An extended version has been accepted for publication in the journal *Social Network Analysis and Mining*, and is available online as a preprint [HLM+16].

10.1 CENTRALITY OF EDGES

The core idea of the research presented here is that not all edges are equally important with respect to properties of a network: For example, a relatively small fraction of long-range edges typically act as shortcuts and are responsible for the small-world phenomenon in complex networks. From a network science perspective, quantifying these differences can yield valuable insights into the importance of relationships and the participating nodes.

Analogously to centralities on nodes (Def. 12), we can differentiate and rank edges by their position within the network's structure. We refer to methods enabling such a ranking as *edge centrality measures*, and the values on which a ranking is based as *edge (centrality) scores*.

Definition 29 (Edge Centrality Measure)**.** Given an undirected graph $G = (V, E)$, an edge centrality measure is a function $c : E \to \mathcal{A}$ which assigns to each edge $\{u, v\}$ an

attribute value $x \in \mathcal{A}$ of (at least) ordinal scale of measurement. The assigned value depends on the position of the edge within the network G as defined by a set of edges $E' \subseteq E$. □

Edge centralities can be defined analogously for directed edges, but we only consider undirected graphs in the following. As a first example, compare the definition of node betweenness (Def. 17) with its analog for edges:

Definition 30 (Edge Betweenness)**.** Naming $|\sigma_{st}|$ the number of shortest paths from a node s to a node t and $|\sigma_{st}(u)|$ the number of shortest paths from s to t that go through the edge $\{u, v\}$, edge betweenness is defined as:

$$c_b(u,v) : \begin{cases} E \to \mathbb{R}_{\geq 0} \\ \{u,v\} \mapsto \frac{1}{n(n-1)} \sum_{s \neq u \neq t} \frac{|\sigma_{st}(u,v)|}{|\sigma_{st}|} \end{cases} \tag{31}$$

□

10.2 EDGE CENTRALITY MEASURES

This section describes a set of edge centrality measures that have been used in the context of network sparsification, including the existing Jaccard Similarity, Simmelian Backbones and edge ranking based on Algebraic Distances. Further we introduce Edge Forest Fire and Local Degree as novel methods. We also present parallel implementations for all methods, though some of them are only partial parallelizations and most parallelizations are straightforward.

10.2.1 *Random Edge (RE)*

When studying different edge centrality measures, the performance of a random edge ranking is an important baseline. As we shall see, it also performs surprisingly well in the sparsification study in Chapter 11. The method selects edges uniformly at random from the original set such that the desired sparsification ratio is obtained. This is equivalent to scoring edges with values chosen uniformly at random. Naturally this needs $O(m)$ time and can be trivially parallelized.

10.2.2 *Triangle Count*

Triangles play an important role because the presence of a triangle indicates the transitivity of the relationship between the three involved nodes. A high frequency of triangles is a major structural property of social networks. The sociological theory of Simmel [SW50] states that "triads (sets of three actors) are fundamentally different from dyads (sets of two actors) by way of introducing mediating effects." In a friendship network, it is likely for two actors with a high number of common friends to be friends as well. Several of the following methods are based on the triangles edge score $T(u, v)$ that denotes for an edge $\{u, v\}$ the number of triangles it belongs to. $T(u, v)$ is sometimes referred to as the *embeddedness* of an edge, and defined equivalently as the number of mutual neighbors shared by its end nodes [BK14]. The time needed for counting the number of all triangles is $O(m \cdot a)$ [OB14], where a is the graph's arboricity [CN85].

Parallelization. We use a novel parallelized variant of the algorithm introduced by [OB14]. This variant is different from the parallel variant introduced in [ST15] as they need additional overhead in the form of sorting operations or atomic operations for storing local counters which we avoid. Algorithm 8 contains the pseudo-code for our algorithm. The algorithm needs a node ordering. While a smallest-first ordering that is obtained by iteratively removing nodes of minimum degree can guarantee the theoretical running time, simply ordering the nodes by degree is actually faster in practice as noticed by [OB14]. Therefore we use such a simple degree ordering. While $N(u)$ denotes all neighbors of u, $N^+(u)$ denotes the neighbors of the node that are higher in the ordering. Note that when using a smallest-first ordering $|N^+(u)|$ is bounded by a. In contrast to [OB14] we count each triangle three times, which does not increase the asymptotic running time. In each iteration step of the outer loop we encounter each triangle u and the edges incident to u exactly once. Therefore it is enough to count the triangle for the edges that are incident to u and where u has the higher id. This avoids multiple accesses to the same edge by several threads, we therefore do not need any locks or atomic operations. In the same way we could also update triangle counters per node e.g. for computing clustering coefficients without additional work and without using locks or atomic operations. Note that node markers are thread-local.

Algorithm 8: Parallel triangle counting

```
1 foreach u ∈ V do in parallel
2     Mark all v ∈ N(u)
3     foreach v ∈ N(u) do
4         foreach w ∈ N⁺(u) do
5             if w is marked then
6                 Count triangle u, v, w
7     Un-mark all v ∈ N(u)
```

10.2.3 *(Local) Jaccard Similarity (JS, LJS)*

One line of research attempts to sparsify graphs with the goal of speeding up data mining algorithms. [SPR11] propose a local graph sparsification method with the intention of speedup and quality improvement of community detection. It has also been adapted for accelerating *collective classification*, the task of inferring the labels of all nodes in a graph given a subset of labeled nodes [SRD13].

The method uses the Jaccard measure to quantify the overlap between node neighborhoods $N(u)$, $N(v)$ and thereby the (Jaccard) similarity of two given nodes:

$$\mathrm{JS}(u,v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|} = \frac{T(u,v)}{d(u) + d(v) - T(u,v)} \tag{32}$$

where $d(u)$ denotes the degree of u. Clearly $JS(u,v)$ also serves an edge centrality measure. The time needed for calculating the Jaccard Similarity is the time for counting all triangles. The authors also propose a fast approximation which runs in time $O(m)$.

For this Jaccard Similarity, [SPR11] propose the local filtering technique (explained in detail in Sec. 11.2.1). We denote the edge scores derived thusly by LJS. The time needed

for calculating this local edge score is the time for calculating the Jaccard Similarity and for sorting the neighbors of all nodes, which can be done in $O(m \log(d_{\max}))$. We process the nodes in parallel for sorting the neighbors.

Parallelization: With our parallel triangle counting variant and pre-calculated node degrees, the edge scores for Jaccard Similarity can be calculated in parallel. We are not aware of any prior work that calculated the Jaccard Similarity in parallel. As shown in Algorithm 10 in Chapter 11 also the local filtering can be parallelized. As we will show in our experimental evaluation the achieved running times with our parallel implementation are very good and not far from random edge filtering.

10.2.4 *Simmelian Backbones (TS, QLS)*

The *Simmelian Backbones* introduced by Nick et al. [NLCB13] aim at discriminating between edges that are placed within dense subgraphs and those between them. The original goal of these methods was to produce readable layouts of networks. To achieve a "local assessment of the level of actor neighborhoods" [NLCB13], the authors propose the following approach, which we adapt to our concept of edge scores. Given an edge centrality measure S and a node u, they introduce the notion of a rank-ordered neighborhood as the list of adjacent neighbors sorted by $c(u, \cdot)$ in descending order. The original *(Triadic) Simmelian Backbone* uses triangle counts T for c. The newer *Quadrilateral Simmelian Backbone* by Nocaj et al. [NOB14] uses *quadrilateral edge embeddedness*, which they define as

$$Q(u, v) = \frac{q(u, v)}{\sqrt{q(u) \cdot q(v)}} \tag{33}$$

with $q(u, v)$ being the number of quadrangles containing edge $\{u, v\}$ and $q(u)$ being the sum of $q(u, v)$ over all neighbors v of u. They argue that this modified version performs even better at discriminating edges within and between dense subgraphs.

On top of the rank-ordered neighborhood graph that is induced by the ranked neighborhoods of all nodes, Nick et al. introduce two filtering techniques, a parametric one and a non-parametric one. Like Nocaj et al. we use only the non-parametric variant. By *TS*, we denote the Triadic Simmelian Backbone and by *QLS* the Quadrilateral Simmelian Backbone. The non-parametric variant uses the Jaccard measure similar to Local Similarity but, instead of considering the whole neighborhood, they use the maximum of the Jaccard measure of the top-k neighborhoods for all possible values of k. While the time needed for quadrangle counting is equal to the time for triangle counting [CN85], the overlap and Jaccard measure calculation of prefixes needs time $O(m \cdot d_{\max})$ as it needs to be separately calculated for all edges. We use a relatively simple implementation of the original algorithm for quadrangle counting. All neighborhoods are sorted in parallel which takes $O(m \cdot \log(d_{\max}))$ time. By using binary vectors for marking the unmatched neighbors of both incident nodes we get $O(\sum_{\{u,v\} \in E} d(u) + d(v)) = O(m \cdot d_{\max})$ for the Jaccard measure calculations which dominates the running time. We execute this calculation in parallel for all edges.

Parallelization: Our implementation of triangular Simmelian Backbones is fully parallelized, we use our parallel triangle counting implementation, then we sort all neighborhoods in parallel. Using this information we can compute the triangular Simmelian Backbone scores in parallel for all edges. For the quadrilateral Simmelian Backbones, the

quadrangle counting step is sequential as we are not aware of an efficient, parallelized quadrangle counting algorithm on edge level but the remaining part of the algorithm is parallelized as in the case of triangular Simmelian Backbones.

10.2.5 *Edge Forest Fire (EFF)*

The original Forest Fire node sampling algorithm [LF06] is based on the idea that nodes are "burned" during a fire that starts at a random node and may spread to the neighbors of a burning node. Note that contrary to random walks the fire can spread to more than one neighbor but already burned neighbors cannot be burned again. The basic intuition is that nodes and edges that get visited more frequently than others during these walks are more important. In order to select edges instead of nodes, we introduce a variant of the algorithm in which we use the frequency of visits of each edge as a proxy for its relevance.

Algorithm 9 shows the details of the algorithm we use to compute the edge score. The fire starts at a random node which is added to a queue. The fire always continues at the next extracted node v from the queue and spreads to neighboring unburned nodes until either all neighbors have been burned or a random probability we draw is above a given burning probability threshold p. The number of burned neighbors thus follows a geometric distribution with mean $p/(1-p)$. As the total length of all walks is hard to estimate in advance, we cannot give a tight bound for the running time.

Algorithm 9: Edge Forest Fire

```
Input: targetBurnRatio ∈ ℝ, p ∈ [0, 1)
edgesBurnt ← 0
while edgesBurnt < m· targetBurnRatio do
    Add random node to queue
    while queue not empty do
        v ← node from queue
        while true do
            q ← random element from [0, 1)
            if q > p or v has no un-burnt neighbors then
                break
            x ← random un-burnt neighbor of v
            Mark x as burnt
            Add x to queue
            Increase edgesBurnt
            Increase burn counter of {v, x}
```

Parallelization: We use a very simple parallelization for our Edge Forest Fire algorithm. We burn several fires in parallel with separate burn markers per thread and atomic updates of the burn frequency. In order to avoid too frequent updates, we update the global counter for the number of edges burned only after a fire has stopped burning before we start the next fire.

10.2.6 *Algebraic Distance (AD)*

Algebraic distance [CS11] (α) is a method for quantifying the structural distance of two nodes u and v in graphs, described in Sec. 2.2. In a straightforward way, algebraic distance can be used to quantify the "range" of edges, with short-range edges (low $\alpha(u, v)$ for an edge $\{u, v\}$) connecting nodes within the same dense subgraph, and long-range edges (high $\alpha(u, v)$ for an edge $\{u, v\}$) forming bridges between separate regions of the graph. Hence, α restricted to the set of connected node pairs is an edge score in our terms, and can be used to filter out long- or short-range edges. We use $1 - \alpha(u, v)$ as edge score in order to treat short-range edges as important. As parameters for the algebraic distance computation, we choose 20 systems and 20 iterations.

Parallelization: Updates of node coordinates can be easily executed in parallel for all nodes as the new values only depend on the values of the previous round. The implementation is described in detail in Sec. 5.1.1.

10.2.7 *Local Degree (LD)*

Inspired by the notion of hub nodes, nodes with relatively high degree, we propose a new measure for the local importance of edges. The basic idea, developed in the context of sparsification, is that we want to retain (and therefore rank highly) such edges that lead from a node v to a neighbor that has high degree among all neighbors of v. For each node $v \in V$, we want to include the edges to the top $\lfloor \deg(v)^{\alpha} \rfloor$ neighbors, sorted by degree in descending order. This idea can be mapped to an edge score: We use $1 - \alpha$ for the minimum parameter α such that an edge would be contained in a sparsified graph. (For formal details, see the definitions in Sec. 11.2.1). The goal of this approach is to keep those edges in the sparsified graph that the edges ranked highly by this method form what can be considered a "hub backbone" of the network, emphasizing edges leading to (local or global) hubs. Such nodes with relatively high degree have been found to significantly influence a complex network's topology. This idea is examined in more detail in Sec. 10.3.1.

As only the neighbors of each node need to be sorted, this can be done in $O(m \log(d_{\max}))$. Using linear-time sorting it is even possible in $O(m)$ time. We have decided against the linear-time variant and instead apply the sorting in parallel on all nodes.

Parallelization: The parallelization of the Local Degree score calculation is very similar to the parallelization for local filtering (described in more detail in Ch. 11) We can handle all nodes in parallel. For each node we can independently sort the neighbors and assign the appropriate scores. Again we need to use atomic maximum calculation in order to make sure that parallel updates of edge scores are handled correctly.

10.3 EXPERIMENTAL STUDY

Our experiments have been performed on a multicore compute server with 4 physical Intel Core i7 cores at 3.4 GHz, 8 threads, and 32 GB of memory. For this explorative study, we use a collection of 100 social networks representing early snapshots of Facebook, each of which is an online friendship network for a US university or college [TMP12], most members are students. Sizes of the Facebook networks are between 10k and 1.6 million

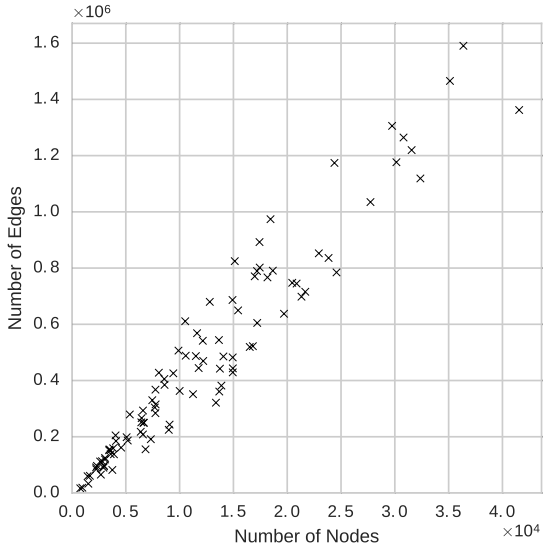


Figure 55: Number of nodes and edges of the Facebook networks

edges, the number of nodes and edges is shown in Fig. 55. We use implementations described in more detail in Sec. 11.3.

10.3.1 *Understanding Local Degree Scores*

The novel Local Degree edge rating method we propose merits a closer look in order to better understand its outcome and effectiveness, particularly in light of its good performance in the following study on sparsification (see Sec. 11.4). Starting from a very simple local principle ("edges $\{u, v\}$ incident to node u are the more important the higher the degree of v"), what is – globally viewed – the edge structure highlighted by the measure? Its initial motivating idea was to conjecture that networks contain a "hub backbone", a subgraph of connected nodes with high degree relative to their neighbors. Because hubs connect different regions of the network, such a hub backbone would be significantly responsible for important properties of the network, such as the small-world phenomenon. Consequently, edges connected with hub nodes are significantly more likely to be part of shortest paths. In the following we briefly examine this idea empirically, reporting edge rank correlation coefficients for edge betweenness (Def. 30) and Local Degree scores in Table 9 for a small set of networks from different contexts. The correlation can vary widely among networks of different types, but is often positive, especially for social networks (`fb-...`, `PGPgiantcompo`, `jazz`). In networks known to have similar structure, the $O(m)$ time Local Degree edge scoring method can serve as a very rough heuristic to find edges with high betweenness without performing computationally expensive shortest path calculations for all node pairs. A strong correlation of this kind contributes to the observed behavior when applying Local Degree scores for sparsification, i.e. preserving the network diameter and shortest-path-based relative centralities such as betweenness. The performance of the Local Degree method is further discussed with the interpretation of results of the sparsification study in Chapter 11.

network	type	Spearman's ρ
dolphins	animal social network	0.32
power	power grid topology	0.14
as-22july06	internet topology	0.64
PGPgiantcompo	PGP web of trust	0.64
fb-MIT8	Facebook social network	0.57
fb-Smith60	Facebook social network	0.57
fb-Caltech36	Facebook social network	0.63
jazz	musicians collaborations	0.57
mouse_brain	anatomical connectome	0.05
foodweb-baydry	food web	0.16

Table 9: Correlation between Local Degree edge scores and edge betweenness

10.3.2 *Correlations between Edge Scores*

Among the edge centrality measures considered here, some are more similar to others in the sense that they tend to produce similar edge rankings. Such similarities can be clarified by studying correlations between edge scores. We calculate edge score correlations for the set of 100 Facebook networks as follows: For each single network, edge scores are calculated with the various scoring methods and Spearman's rank correlation coefficient is applied. The coefficient is then averaged over all networks and plotted in the correlation matrix (Fig. 56). There is one column for each method, and the column Mod represents edge scores that are 1 for intra-community edges and 0 for inter-community edges after running a modularity-maximizing Louvain community detection algorithm. Positive correlations with these scores indicate that the respective rating method assigns high scores to edges within modularity-based communities. The column Tri simply represents the number of triangles an edge is part of. As some of the methods are normalizations of this score, this shows how similar the ranking still is to the original score.

Interpretation of the results is challenging: The correlations we observe reflect intrinsic, mathematical similarities of the rating algorithms on the one hand, but on the other hand they are also caused by the structure of this specific set of social networks (e.g., it may be a characteristic of a given network that edges leading to high-degree nodes are also embedded in many triangles). Nonetheless, we note the following observations: There are several groups of methods. Simmelian Backbones, Jaccard Similarity and Triangles are highly positively correlated, which is not unexpected as they are all based on triangles or quadrangles and are intended to preserve dense subgraphs. Algebraic distance is still positively correlated with these methods but not as strongly even though it is also intended to prefer dense subgraphs. An explanation for this weaker correlation is that, while both prefer dense regions, the order of the individual edges is different. All of the previously mentioned methods are also correlated with the Modularity value, algebraic distance with local filtering has the highest score among all of these methods. Our experiments on the preservation of community structure (Sec. 11.4.2) confirm this relationship. The correlation of the Modularity value and these methods are similar to the correlation between algebraic distance and the rest of the methods which shows again

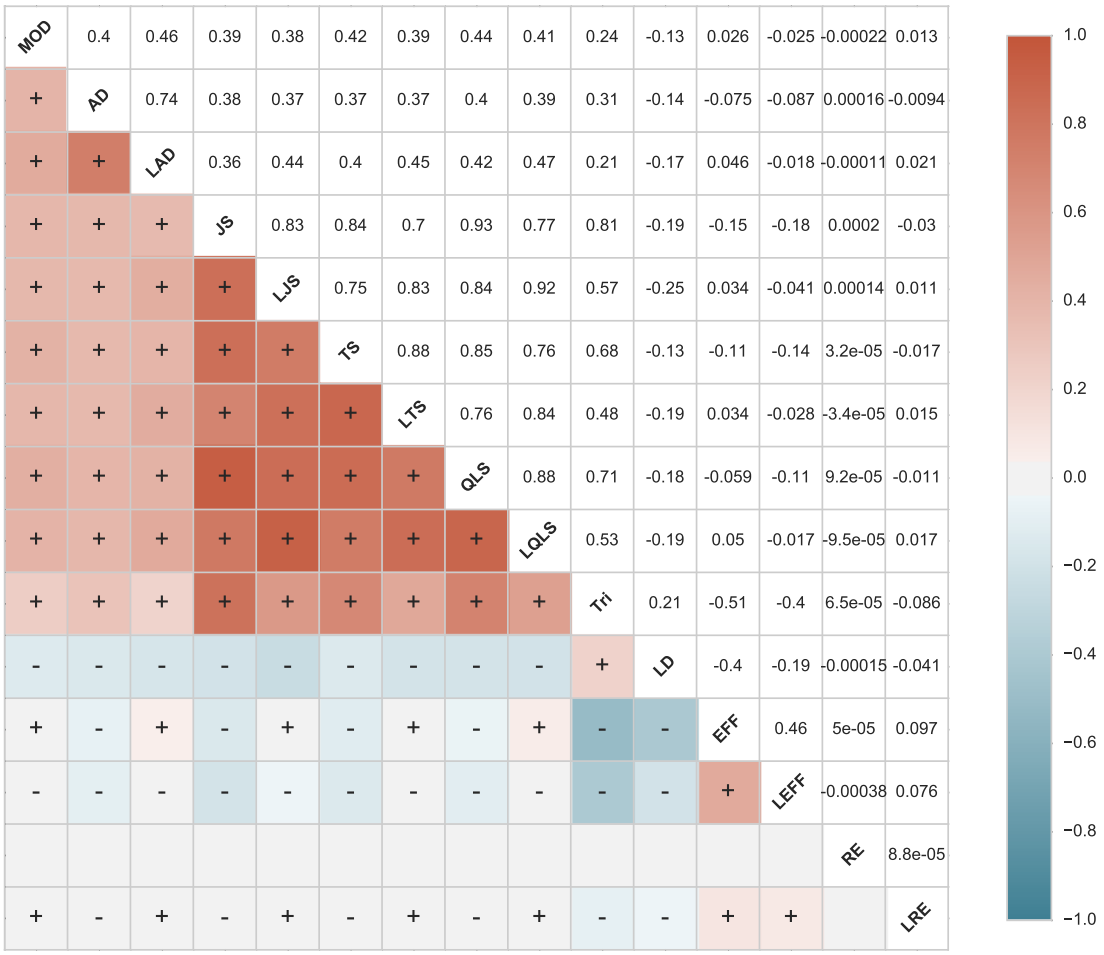


Figure 56: Edge score correlations (Spearman's ρ, average over 100 Facebook networks)

that the lower correlation values are probably due to different orderings of the individual edges.

Our new method Local Degree is slightly negatively correlated with all these methods but still positively correlated with the Triangles. It is also slightly negatively correlated with the Modularity value, this is due to the method's preference of inter-cluster edges, which is also confirmed by our experiments below. The newly introduced Edge Forest Fire is also negatively correlated with Local Degree and even more negatively with Triangles. This strong negative correlation between Edge Forest Fire and triangle count can be explained by the fact that the Edge Forest Fire can never "burn" a triangle, as nodes cannot be visited twice. Random edge filtering is not correlated at all, which is definitely expected.

It is interesting to see that each method is also relatively strongly correlated with its local variant (discussed in detail in Ch. 11), apart from random edge filtering (we use different random values as basis of the local filtering process). Even the Edge Forest Fire method, which should also be relatively random, has a positive correlation with its local variant. This shows that it prefers a certain kind of edge and that this preference is kept when applying the local filtering.

Among the variants of Simmelian Backbones and Jaccard Similarity also the local variants are more correlated to other local variants than to other non-local variants and also not as strongly correlated to triangles. This shows that the local filtering indeed adds another level of normalization. Also Jaccard Similarity seems to be more correlated to Quadrilateral Simmelian Backbones than to the variant based on triangles even though Jaccard Similarity is based on triangles itself. This is also interesting to see, as Quadrilateral Simmelian Backbones are computationally more expensive than the Jaccard Similarity.

10.3.3 *Running Time*

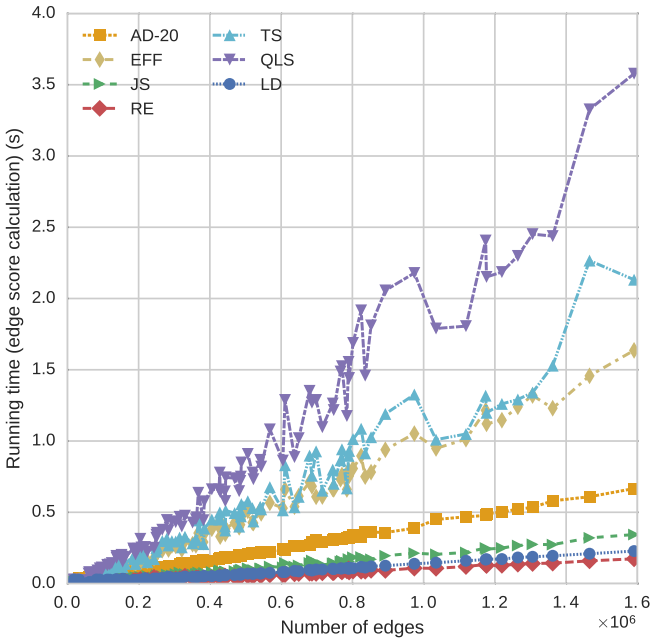


Figure 57: Running times of various edge scoring methods on the Facebook networks

Measured running times are shown in Fig. 57. Random Edge selection is clearly the fastest method, closely followed by Local Degree. Jaccard Similarity is also not much slower and scales also very well. Therefore these methods are well suited for large-scale networks in the range of millions to billions of edges. The efficiency of the Jaccard Similarity method shows that our parallel triangle counting implementation is indeed very scalable. The authors also proposed inexact Jaccard coefficient calculation for a further speedup though given our numbers it can be doubted if – given an efficient triangle counter – this is necessary or helpful at all. Algebraic distance is a bit slower but scales very well nevertheless. Using less systems or iterations could further speed-up algebraic distance if speed is an issue. Both Simmelian methods are significantly slower than the other methods, but still efficient enough for the network sizes we consider. The visible difference between quadrilateral and triangular Simmelian Backbones can be explained by the difference between triangle and quadrangle counting, additionally we

did not parallelize the latter. While the time complexity in O-notation of Edge Forest Fire is difficult to assess, it seems to be slightly faster than Simmelian Backbones.

11

SPARSIFICATION OF SOCIAL NETWORKS

> The old scribe Qi was tasked with assembling and maintaining the high-level design document for a new system. Each developer provided the scribe with his or her contributions, but the contributions of the head monk were returned with the subject line "Unsatisfactory".
>
> "What is your objection?" demanded the head monk, who was known for his impatience. "Much of what you gave me was actual source code," said the scribe. "Class declarations, method bodies, long SQL queries. That is not our way." "How better to document the algorithms?" asked the head monk pointedly. "My way distorts nothing." The scribe considered this. "Very well," he said finally. "If you can convince the head priest of my order, I will allow it." "Tell me where this priest may be found," said the head monk, fumbling in his robes for paper and ink.
>
> "I can only tell you where I found him last, about three years ago," said the scribe, thumbing through his diary. "Ah! Here it is: I followed the Road of White Nettles toward the East for two days, until I came to a stream whose banks were swollen from a recent rain. There I turned so the wind was at my back, walking until two salmon leaped from the waters. Crossing at the next footbridge I took the fork that pointed straight at the moon, turned right when I came to a barefoot boy gathering sticks and then left when the clouds blotted out the pole-star. It was the hut with the door standing halfway open; you can't miss it."
>
> The head monk required no further correction.
>
> – from *The Codeless Code*

Sparsification reduces the size of networks while preserving structural properties of interest. Various sparsifying algorithms have been proposed in different contexts. We contribute the first systematic conceptual and experimental comparison of *edge sparsification* methods on a diverse set of network properties. It is shown that they can be understood as methods for rating edges by importance and then filtering globally or locally by these scores. We propose a local filtering step that has been introduced by [SPR11] for one specific sparsification technique as a generally applicable and beneficial post-processing step for preserving the connectivity of the network and most properties we consider. We show that applying this local filtering technique improves the preservation of all kinds of properties. In addition, we propose a new sparsification method (*Local Degree*) which preserves edges leading to local hub nodes. All methods are evaluated on a set of social networks from Facebook, Google+, Twitter and LiveJournal with respect to network properties including diameter, connected components, community structure, multiple node centrality measures and the behavior of epidemic simulations. Experiments with our implementations of the sparsification methods show that many network properties can be preserved down to about 20% of the original set of edges for sparse graphs with a reasonable density. The experimental results allow us to differentiate the behavior of different methods and show which method is suitable with respect to which property. While our Local Degree method is best for preserving connectivity and short distances, other local variants we introduce are best for preserving the community structure.

11.1 INTRODUCTION

11.1.1 *Context*

Most real-world complex networks, including social networks, are already sparse in the sense that for n nodes the edge count m is asymptotically in $O(n)$. Nonetheless, typical densities lead to a computationally challenging number of edges. Here we pursue the goal of further sparsifying such networks by retaining just a fraction of edges (sometimes called a "backbone" of the network), while showing experimentally that important properties of networks can be preserved in the process.

Potential applications of network sparsification are numerous. One of them is information visualization: Even moderately sized networks turn into "hairballs" when drawn with standard techniques, as the amount of edges is visually overwhelming. In contrast, showing only a fraction of edges can reveal network structures to the human eye if these edges are selected appropriately. Sparsification can also be applied as an acceleration technique: By disregarding a large fraction of edges that are unimportant for the task, running times of graph and network analysis algorithms can be reduced. Many other possible applications arise if we think of sparsification as lossy compression. Large networks can be strongly reduced in size if we are only interested in certain structural aspects that are preserved by the sparsification method.

Despite the similar terminology, our work is only weakly related to a line of research in theoretical computer science where *graph sparsification* is understood as the reduction of a dense graph ($\Theta(n^2)$ edges) to a sparse ($O(n)$ edges) or nearly-sparse graph while provably preserving properties such as spectral properties [BSST13]. The networks of our interest are already sparse in this sense. With the goal of reducing network data size while keeping important properties, our research is related to a body of work that considers sampling from networks (on which [ANK14] provides an extensive overview). Sampling is concerned with the design of algorithms that select edges and/or nodes from a network. Here, node and edge sampling methods must be distinguished: For node sampling, nodes *and* edges from the original network are discarded, while edge sampling preserves all nodes and reduces the number of edges only. The literature on node sampling is extensive, while pure edge sampling and filtering techniques have not been considered as often. A seminal paper [LF06] concludes that node sampling techniques are preferable, but considers few edge sampling techniques. The study presented in [EHR+08] looks at how well a sample of 5%-20% of the original network preserves certain properties, and is mainly focused on node sampling through graph exploration. It concludes that random walk-based node sampling works best on complex networks, but does so on the basis of experiments on synthetic graphs only and compares only with very simple edge sampling methods.

Only edge sampling techniques are directly comparable to our edge scoring and filtering methods. In this work, we restrict ourselves to reducing the edge set, while keeping all nodes of the original graph. Preserving the nodes allows us to infer properties of each node of the original graph. This is important because in network analysis, the unit of analysis is often the individual node, e.g. when a score for each user in an online social network scenario shall be computed. With respect to the goal of accelerating the analysis, many relevant graph algorithms scale with m so reducing m is more relevant.

Another related approach is the *Multiscale Backbone* [SBV09], which is applicable on weighted graphs only and is therefore not included in our study. Instead of applying a global edge weight cutoff for edge filtering, which hides important structures at different scales, this approach aims at preserving them at all scales.

Recently, John and Safro published a study that follows an experimental design similar to that described in the following [JS16]. The focus is on algebraic distance-based sparsification, experimenting with retaining long- or short-range edges, or a mix of the two, and applying sparsification on multiple levels of a hierarchy of coarsened graphs.

11.2 EDGE SPARSIFICATION

All edge sparsification methods we consider can be split up into two stages: (i) the calculation of a score for each of the edges in the input graph (where the score is high if the edge is important) and (ii) subsequent filtering according to such an edge score. In this section we will first define this framework of scoring and filtering edges more formally. For the following formalisms, recall the definition of an edge centrality measure (Def. 29). Using this framework we introduce local filtering as an optionally applicable step. We conclude this section by addressing some limits of edge sparsification.

11.2.1 *Global and Local Filtering*

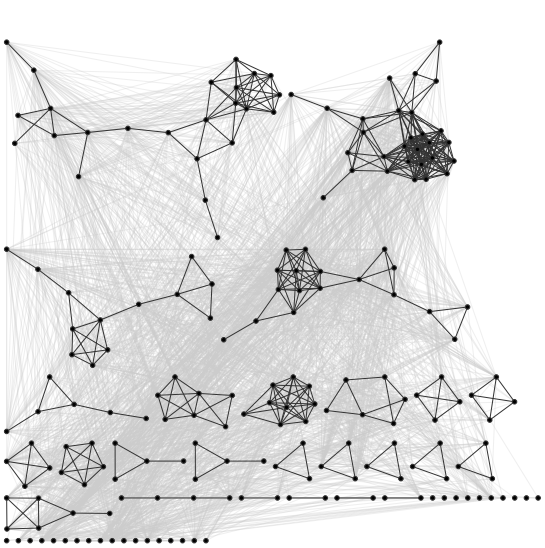
The simplest way to filter by such an edge score is to apply a global threshold and keep all edges whose score is equal to or above the threshold. For the comparison of the different sparsification methods we need to be able to filter the network such that all methods keep an equal percentage of the edges of the network. As the methods produce scores with different ranges of values and different distributions of these values, we cannot simply define a threshold that is the same for all methods. Also using a different threshold per method does not solve the problem as it is possible that many edges have the same score and there is therefore no threshold that leads to the desired ratio of kept edges. Therefore we define global filtering not to apply a given threshold but to filter the edges such that only a given ratio of edges remains:

Definition 31 (Global Filtering)**.** Given a graph $G = (V, E)$ and an edge score $c : E \to \mathbb{R}^+$, a global filtering step by ratio $r \in [0, 1]$ reduces the number of edges in the sparsified graph $G' = (V, E')$ to $\lfloor r \cdot |E| \rfloor$ edges with the highest values of $c(e)$, i.e. $E' \subseteq E$, $|E'| = \lfloor r \cdot |E| \rfloor$ and $\forall e' \in E' \, \forall e \in E \setminus E' : c(e') \geq c(e)$. □

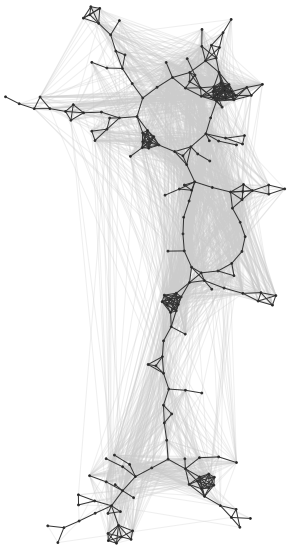
For an actual implementation of global filtering, it is enough to sort all edges by the edge score in descending order and keep the first $\lfloor r \cdot |E| \rfloor$ edges. In order to make sure that in the case of edges with equal score a given order in the graph does not influence our results, we sort edges that have an equal score in a random order by using a random edge score as tie breaker in comparisons.

A problem with global filtering as described above is that methods that are based on local measures like the number of quadrangles an edge is part of tend to assign different scores in different parts of the network as some parts of the network are much denser than other parts. Sparsification techniques like (Quadrilateral) Simmelian Backbones [NOB14] use different kinds of normalizations of quadrangles (see previous section for details) in

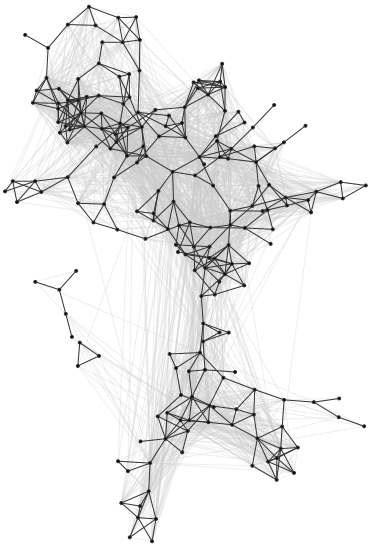
order to compensate for such differences. Unfortunately, these normalizations still do not fully compensate for these differences. In Fig. 58a we visualize a Jazz musicians collaboration network [GD03] with 15% kept edges as an example. As one can see in the figure, many nodes are isolated or split into small components, the original structure of the network (shown with gray edges) is not preserved.



(a) Quadrilateral Simmelian Backbone



(b) Quadrilateral Simmelian Backbone with UMST



(c) Quadrilateral Simmelian Backbone with local filtering

Figure 58: Drawing of the Jazz musicians collaboration network according to a variant of the Quadrilateral Simmelian Backbone with 15% of the edges (in black). Created using *visone* (http://visone.info).

Simmelian Backbones have been introduced for visualizing networks that are otherwise hard to layout. For layouts it is important to keep the connectivity of the network as otherwise nodes cannot be positioned relative to their neighbors. In order to preserve the connectivity, Nocal et al. [NOB14] keep the union of all maximum spanning trees

(UMST) in addition to the original edges. In Fig. 58b we show the result when we keep the UMST. While the network is obviously connected, much of the local structure is lost in the areas between the dense parts – which is not surprising as we only added the union of some trees.

Sataluri et al. [SPR11] face a similar problem, as with their sparsification technique based on Jaccard Similarity (see below for details) they want to preserve the community structure. They propose a different solution: Each node u keeps the top $\lfloor d(u)^\alpha \rfloor$ edges incident to u, ranked according to their similarity where $d(u)$ is the degree of u and $\alpha \in [0,1]$. An edge is kept when it is ranked high enough by one of the nodes that are incident to it. This procedure ensures that at least one incident edge of each node is retained and thus prevents completely isolated nodes.

In order to define this filtering step formally, we first introduce a formal definition of the rank of node v in the neighborhood of node u:

Definition 32 (Neighborhood Rank)**.** Given a graph $G = (V, E)$ and an edge score $c : E \to \mathbb{R}^+$, the neighborhood-rank $r_c(u, v)$ is the position of v in the sorted list of neighbors of u, i.e. $r_c(u, v) := |\{x \in N(u) : c(\{u, x\}) < c(\{u, v\})\}| + 1$. □

Note that when two edges that are incident to u have the same edge score they also have the same rank. Again, we want to be able to keep an exact ratio of edges. In order to achieve this we transform this local rank into a score that can be used for global filtering.

Definition 33 (Local Filtering Score)**.** Given a graph $G = (V, E)$ and an edge score $c : E \to \mathbb{R}^+$, the directed local filtering score $l_c : V \times V \to [0, 1]$

$$l_c(u, v) := \begin{cases} 1, & \text{if } d(u) = 1 \\ 1 - \log(r_c(u, v))/\log(d(u)), & \text{otherwise} \end{cases}$$

Then $l_c(\{u, v\}) := \max(l(u, v), l(v, u))$ is the local filtering score that belongs to c.

□

Lemma 2. *Filtering locally according to an edge score c with parameter α is equivalent to keeping all edges with $l_c(\{u, v\}) \geq 1 - \alpha$.*

Proof. An edge $\{u, v\}$ is kept by local filtering exactly if $r_c(u, v) \leq d(u)^\alpha$ or $r_c(v, u) \leq d(v)^\alpha$, as the top $d(u)^\alpha$ (or $d(v)^\alpha$) edges are kept. If $d(u) = 1$, the only edge that is incident to u is always kept as $1^\alpha = 1$. For $d(u) > 1$, it holds that

$$\begin{aligned} & r_c(u, v) \leq d(u)^\alpha \\ \Leftrightarrow\ & \log(r_c(u, v)) \leq \log(d(u)) \cdot \alpha \\ \Leftrightarrow\ & \frac{\log(r_c(u, v))}{\log(d(u))} \leq \alpha \\ \Leftrightarrow\ & \underbrace{1 - \frac{\log(r_c(u, v))}{\log(d(u))}}_{= l_c(u,v)} \geq 1 - \alpha \end{aligned}$$

Which shows the claim, as $l_c(\{u, v\}$ is exactly defined as the maximum of $l_c(u, v)$ and $l_c(v, u)$, which means that global filtering by $l_c(\{u, v\})$ will keep the edge exactly if one of the directions would have been kept by local filtering. □

In Algorithm 10 we show the parallel algorithm that we use in order to transform edge scores for local filtering. We iterate over all nodes in parallel, sort the neighbors, determine the rank for each neighbor and assign its score. Since the two scores of an edge are possibly calculated at the same time, we use an atomic maximum for assigning the final score. The total time complexity is $O(m \cdot \log(d_{\max}))$, where $d_{\max}$ denotes the maximum degree in G, as every list of neighbors needs to be sorted.

Algorithm 10: Transformation of global edge scores into edge scores that are equivalent to local filtering

```
Input: Graph G = (V, E), edge score s : E → ℝ
Output: Edge score l : E → [0, 1]
l(u, v) ← 0 ∀{u, v} ∈ E
foreach u ∈ V do in parallel
    r ← 0 // rank
    s ← 1 // number of equal scores in a row
    o ← −∞ // old score
    foreach v ∈ N(u) sorted by s(u, v) in descending order do
        if s(u, v) ≠ o then
            r ← r + s
            s ← 1
        else
            s ← s + 1
        if d(u) > 1 then
            e ← 1 − log(r)/ log(d(u))
        else
            e ← 1
        l(u, v) ← atomic max of l(u, v) and e
```

In Fig. 58c we show the Jazz musicians collaboration network, sparsified again to 15% of the edges with the Quadrilateral Simmelian Backbone method using local filtering. With the local filtering step, the network is almost fully connected and local structures are maintained, too. Additionally, as already Satuluri et al. observed when they applied local filtering to their Jaccard Similarity, the edges are much more distributed among the different parts of the network. This means that we can still see the local structure of the network in many parts of the network and do not only maintain very dense but disconnected parts. An explanation for this is that a node u has the minimum degree $d(u)^{\alpha}$ which means high-degree nodes loose more incident edges than low-degree nodes but high-degree nodes still keep more neighbors than low-degree nodes. Therefore some properties are kept but still the differences are smoothened in order to ensure that we retain structure in every part of the network. In our evaluation we confirm that many properties of the considered networks are indeed better preserved when the local filtering step is added. Furthermore, we show that the local filtering step leads to an almost perfect preservation of the connected components on all considered networks even though this is not inherent in the method. This suggests that local filtering is superior to preserving a UMST as not only connectivity but also local structures are preserved.

As one of our contributions we therefore propose to apply this local filtering step to all sparsification methods and not only to Jaccard Similarity, where the local filtering step has been introduced. In our evaluation we do not further consider the alternative of

preserving a UMST as preliminary experiments have shown that adding a UMST has no significant advantage over the local filtering step in terms of the preservation of network properties. With local filtering, our sparsification pipeline consists of the following stages: (i) calculation of an edge score, (ii) conversion of the edge score into a local edge score and (iii) global filtering. In the evaluation we prefix the abbreviations of the local variants with "L".

11.3 IMPLEMENTATION

For this study, we created efficient C++ implementations of all considered sparsification methods, and accelerated them using OpenMP parallelization. In particular, RE, LD, LS and the Simmelian Backbone methods (with exception of the inherently sequential triangle and quadrangle counting algorithms [CN85]) have been parallelized. We implemented the algorithms in *NetworKit* (see Part ii). For community detection, we use the efficient implementation of the Louvain method (PLM), described in Chapter 8. To get consistent results, a deterministic configuration of this algorithm is used.

Gephi [BHJ09] is a graph visualization tool which we use not only for visualization purposes but also for interactive exploration of sparsified graphs. To achieve said interactivity, we implemented a client for the *Gephi Streaming Plugin* in NetworKit. It is designed to stream graph objects from and to Gephi utilizing the JSON format. Using our implementation in NetworKit, a few lines of Python code suffice to sparsify a graph, calculate various network properties, and export it to Gephi for drawing. The approach of separating sparsification into edge score calculation and filtering allows for a high level of interactivity by exporting edge scores from NetworKit to Gephi and dynamic filtering within Gephi.

11.4 EXPERIMENTAL STUDY

Our experimental study consists of two parts. In the first part (Sec. 10.3.2) we compare correlations between the calculated edge scores on a set of networks. In the second part (Sec. 11.4.2) we compare how similar the sparsified networks are to the original network by comparing certain properties of the networks.

11.4.1 *Setup*

The following experimental part uses the same platform and the set of 100 Facebook social networks already described in Sec. 10.3 in the previous chapter. Unless otherwise noted, we aggregate experimental results over this set of networks. The common origin and the high structural similarity among the networks allows us to get meaningful aggregated values. For the experiments on the preservation of properties we also use the Twitter and Google+ networks [LM12] and the LiveJournal (com-lj) network from [YL12]. All of them are friendship networks, the Twitter and Google+ networks consist of the combined ego networks of 973 and 132 users, respectively. In Table 10 we provide the number of nodes and edges as well as diameter and clustering coefficient averaged over all Facebook networks and the individual values for the three other networks. Furthermore we provide the number of edges divided by the number of nodes, which indicates how

network	n	m	m/n	δ	c
Facebook	12 083	469 845	38.4	7.8	0.25
com-lj	3 997 962	34 681 189	8.7	21	0.35
gplus	107 614	12 238 285	113.7	6	0.52
twitter	81 306	1 342 296	16.5	7	0.60

Table 10: Number of nodes n, the number of edges m, m/n, the diameter δ and the average local clustering coefficient c of the used social networks (average values for the Facebook networks)

much redundancy there is in the network. If this number was near 1 and the network connected, the network would be close to a tree in structure. It is not realistic to expect that we can preserve the structure of the network if it is very sparse already. The networks we selected have a varying degree of redundancy but all of them are dense enough such that if we remove 80% of the edges more edges remain than we would need for a single tree. The characteristics of the Facebook networks are relatively similar.

It remains an open question to what extent results can be translated to other types of complex networks, since according to experience the performance of network analysis algorithms depends strongly on the network structure.

11.4.2 *Similarity in Network Properties*

Quantifying the similarity between a network and its sparsified version is an intricate problem. Ideally, a similarity measure should meet the following requirements:

1. *Ignoring trivial differences*: Consider, for example, the degree distribution: One cannot expect the distribution to remain identical after edges get removed during sparsification. It is clear, however, that the general shape of the distribution should remain "similar" and that high-degree nodes should remain high-degree nodes in order to consider the degrees as preserved.

2. *Intuitive and Normalized*: Similarity values from a closed domain like $[0, 1]$ allow for aggregation and comparability. A similarity value of 1 indicates that the property under consideration is fully preserved, whereas a value of 0 indicates that similarity is entirely lost. In some cases we also used relative changes in the interval $[-1, 1]$ where 0 means unchanged as they provide a more detailed view at the changes.

3. *Revealing Method Behavior*: A good similarity measure will clearly expose different behavior between sparsification methods.

4. *Efficiently computable.*

Following these requirements, we select measures that quantify relative changes for important structural properties of a network (as introduced in Chapter 2). While this collection cannot be exhaustive, we focus on common measures, including global properties like diameter, size of the largest connected component and quality of a community

structure. Node degree, betweenness and PageRank are node centrality indices which represent a ranking of nodes by structural importance. Since absolute values of the centrality scores are less interesting than the resulting rank order, we compare the rankings before and after sparsification using Spearman's ρ rank correlation coefficient. (This focus on rank order is also the reason why we did not adopt the Kolmogorov-Smirnov statistic used in [LF06], which compares distributions of absolute values.) Even though the local clustering coefficient can be interpreted as a centrality score as well, the comparison of ranks does not seem meaningful in this case due to the fact that it is a local score. Instead, we analyse the deviation of the average local clustering coefficient from the original value.

In the following plots, the measures are shown on the y-axis for a given ratio of kept edges (m'/m) on the x-axis (e.g., a ratio of 0.2 means that 20% of edges are still present). For each value there are two rows of plots. The first contains averages over the 100 Facebook networks with error bars that indicate the standard deviation. The second row contains the values at 20%, 50% and 80% remaining edges of the three other networks. In each row, we show two plots: the left plot with the non-local methods and the right plot with the methods that use local filtering.

Connected Components

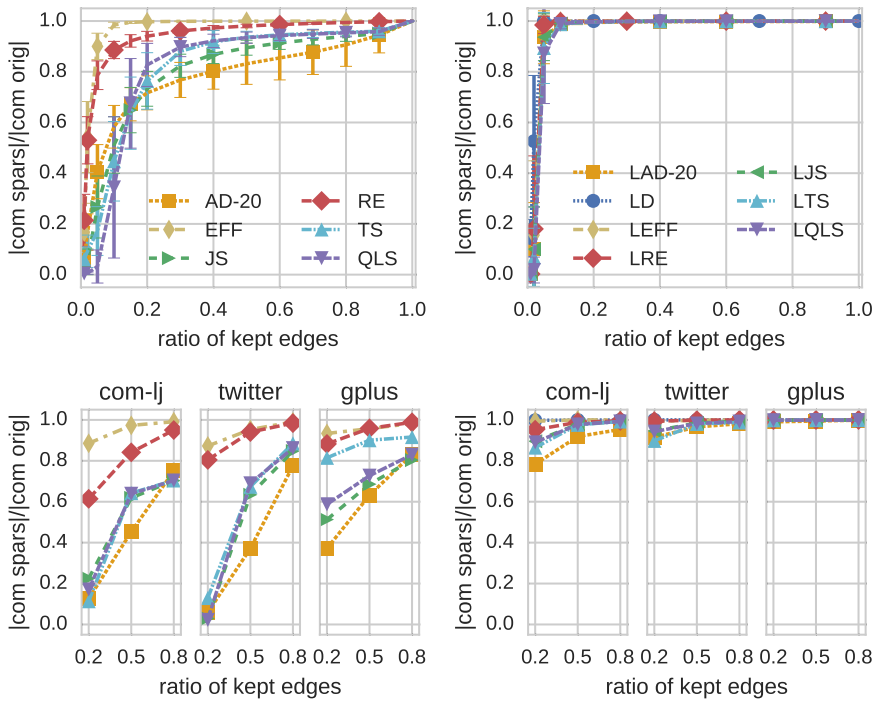
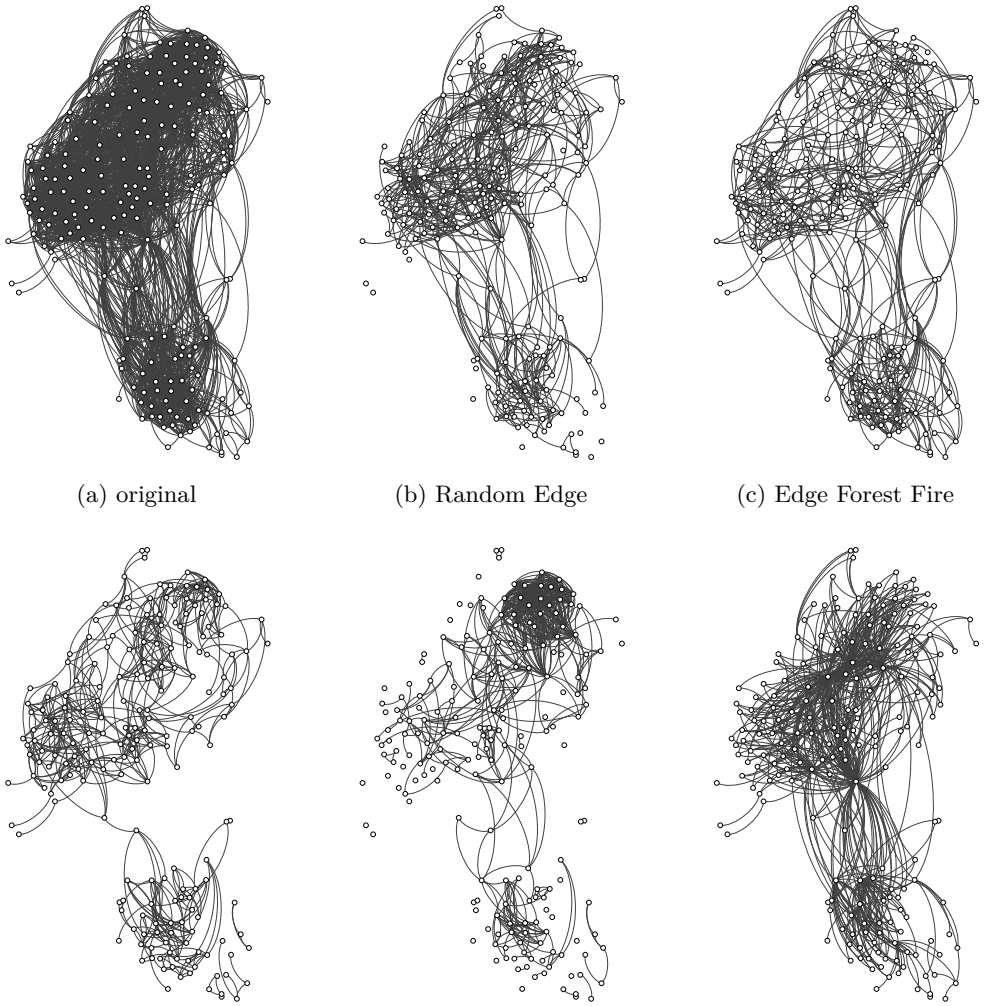


Figure 60: The size of the largest component in the sparsified network divided by the size of the largest component in the original network.

As all of our networks have, like most real-world complex networks, a giant component that comprises most nodes we track its change by dividing the size of the largest component in the sparsified network by the size of the largest component in the original network. As shown in Fig. 60, out of the non-local methods Edge Forest Fire best preserves the connected component. Random edge deletion leads to a slow decrease in the size of the



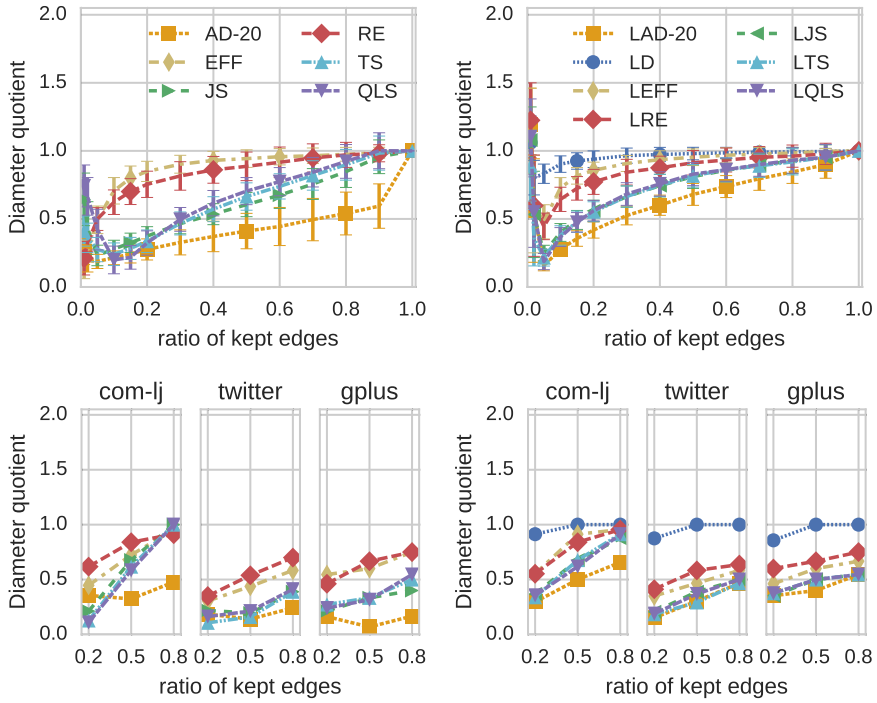
(a) original (b) Random Edge (c) Edge Forest Fire

(d) Local Similarity (e) Triangular Simmelian Backbone (f) Local Degree

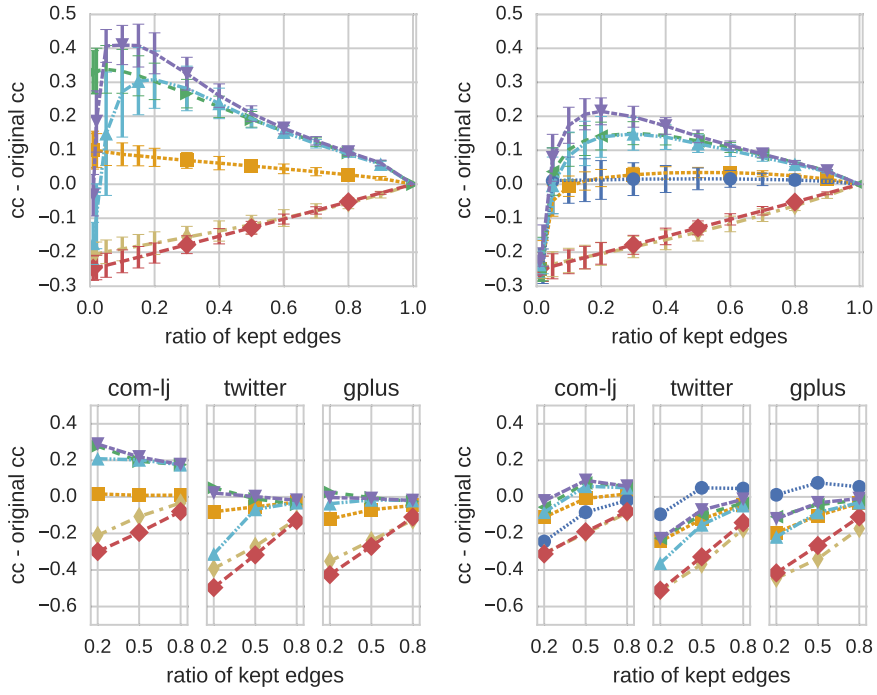
Figure 59: Sparsification methods applied to the Jazz musicians collaboration network: 80 % of edges removed

largest component while Simmelian Backbones, Jaccard Similarity and algebraic distance lead to a separation very quickly. Below 20% of retained edges, the size of the largest component on the Facebook networks drops very quickly, here the networks seem to be decomposed into multiple smaller parts. On the other networks, this drop occurs at different ratios of kept edges which reflects their different densities and probably also their different structures. Local Filtering is able to maintain the connectivity. On the Facebook networks, all methods keep the largest component almost fully connected up to 20% of retained edges, only below that small differences are visible. The results on the LiveJournal, Twitter and Google+ networks show that – as expected – with increasing density it is easier to preserve the connectivity of the network. Our Local Degree method best preserves the connected components of all networks, closely followed by the local variant of random edge deletion and Edge Forest Fire.

Diameter



(a) Original network diameter divided by network diameter



(b) Deviation from original clustering coefficient

Figure 61: Preservation of global network properties

In order to observe how the network diameter changes through sparsification, we plot the quotient of the original network diameter and the resulting diameter, which yields legible results since in practice the diameter is mostly increased during sparsification. We compute the exact diameters using a variation of the ExactSumSweep algorithm [BCH+15].

We motivate the Local Degree method with the idea that shortest paths commonly run through hub nodes in social networks. Therefore, preserving edges leading to high-degree nodes should preserve the small diameter. This is confirmed by our experiments (Fig. 61a). In contrast, methods that prefer edges within dense regions clearly do not preserve the diameter. With Simmelian Backbones the diameter drops when only few edges are left; this can be explained by the fact that Simmelian Backbones do not maintain the connectivity and that at the end the graph is decomposed into multiple connected components which have a smaller diameter. Algebraic distance is even more extreme in this aspect. Local filtering leads to a slightly better preservation of the diameter when applied to the other methods but algebraic distance remains the worst method in this regard. Note that the LiveJournal network has a higher diameter than the other networks (see Table 10); this might explain why the diameter is better preserved there.

Clustering Coefficient

Fig. 61b shows the deviation of the average local clustering from the value of the original network. Both for local and non-local methods we observe three classes of methods on the Facebook networks: methods that clearly decrease the clustering coefficient, methods that preserve the clustering coefficient and methods that increase it.

For both Random Edge and Edge Forest Fire, which are based on randomness, the clustering coefficient drops almost linearly with decreasing sparsification ratio. This can also be observed on the other three networks. The additional local filtering step does not significantly change this.

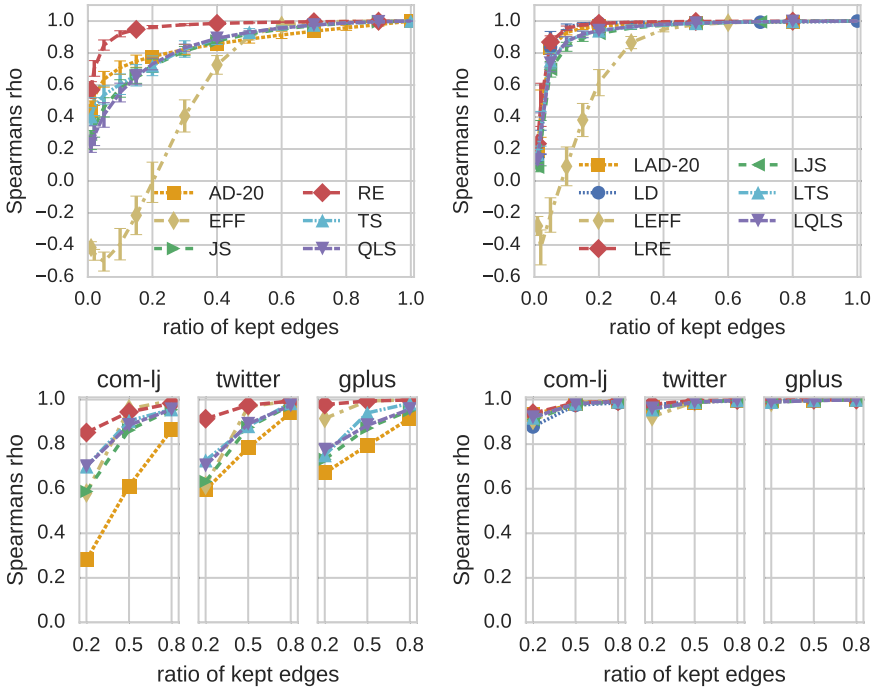
Simmelian Backbones and Jaccard Similarity keep mostly edges within dense regions, which results in increasing clustering coefficients on all networks. Triadic Simmelian Backbones show the weakest increase, on the Twitter network even a decrease of the clustering coefficients. Note that with 0.52 and 0.6 the clustering coefficients are already relatively high on the Google+ and Twitter networks, therefore the very small increase is not surprising. Local filtering slightly weakens this effect on the Facebook networks, on the other networks it is even reversed. Given the high clustering coefficients in the original networks, this is not very surprising as we would need to retain very dense areas while local filtering leads to a more balanced distribution of the edges.

From the previous results especially concerning the connected components one would expect that algebraic distance also increases the clustering coefficients. Interestingly though, filtering using algebraic distance leads to a slight increase of the clustering coefficient on the Facebook networks, constant clustering coefficients on the LiveJournal network and even slightly decreasing clustering coefficients on the Twitter and Google+ networks. With the additional local filtering step algebraic distance almost preserves the clustering coefficients on the Facebook networks while on the other networks it is slightly decreased. Algebraic distance leads to random noise on the individual edge weights, therefore they probably lead to a more random selection of edges that also destroys more triangles than the selection of Simmelian Backbones and Jaccard Similarity. Our Local Degree method best preserves the clustering coefficient on the Facebook networks, though with

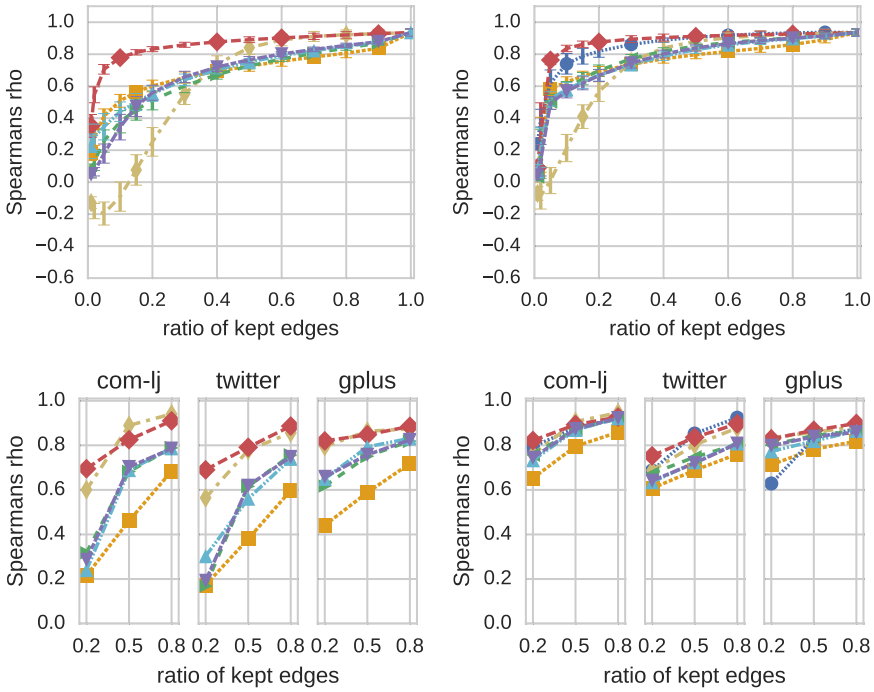
some differences between the various networks in the dataset (note the error bars). On the LiveJournal network it leads to a decrease of the clustering coefficient while on the Twitter and Google+ networks it leads to a slight increase of the clustering coefficient. This is probably due to the special structure of ego networks.

Our experiments therefore do not reveal a general best method for preserving clustering coefficients. If high clustering coefficients shall be created or preserved, the Jaccard Similarity and Quadrilateral Simmelian Backbones seem to be a good choice. Algebraic distance is good at preserving clustering coefficients with slight deviations but our Local Degree method also works well on the considered social networks.

Node Centrality Measures

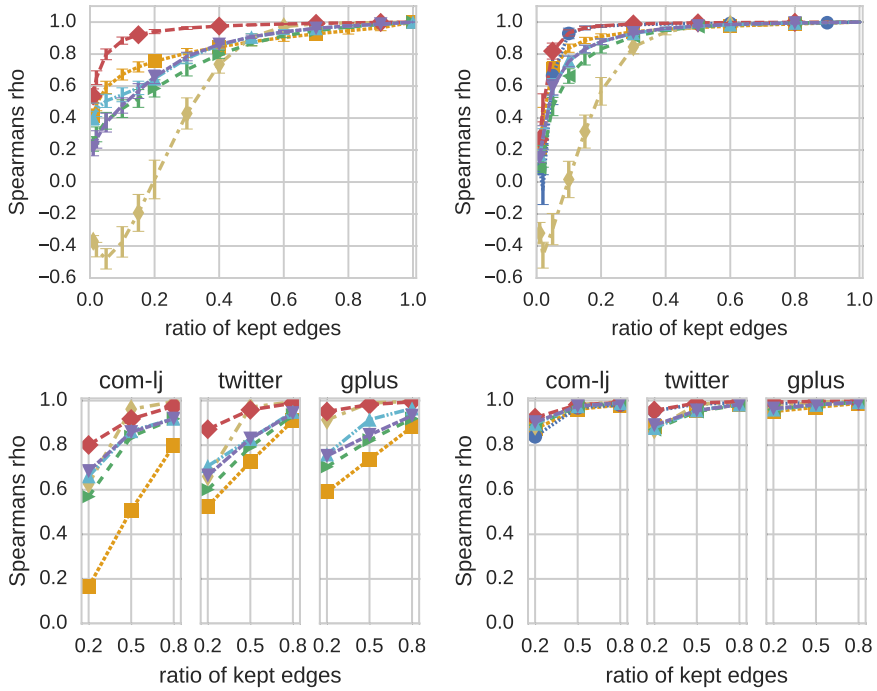


(a) Preservation of node degree



(b) Preservation of betweenness centrality

Figure 62: Preservation of the ranking of node centrality measures (Spearman's ρ rank correlation coefficient)



(c) Preservation of PageRank centrality

Figure 62: (cont.) Preservation of the ranking of node centrality measures (Spearman's ρ rank correlation coefficient)

The exact calculation of betweenness centrality is in practice too expensive for the whole set of networks and sparsification methods we consider. Therefore we use the approximation algorithm [GSS08] with at least 16 samples, for smaller networks also with up to 512 samples. For the calculation of the PageRank centrality we use a damping factor of 0.85 and an error tolerance of 10^{-9}.

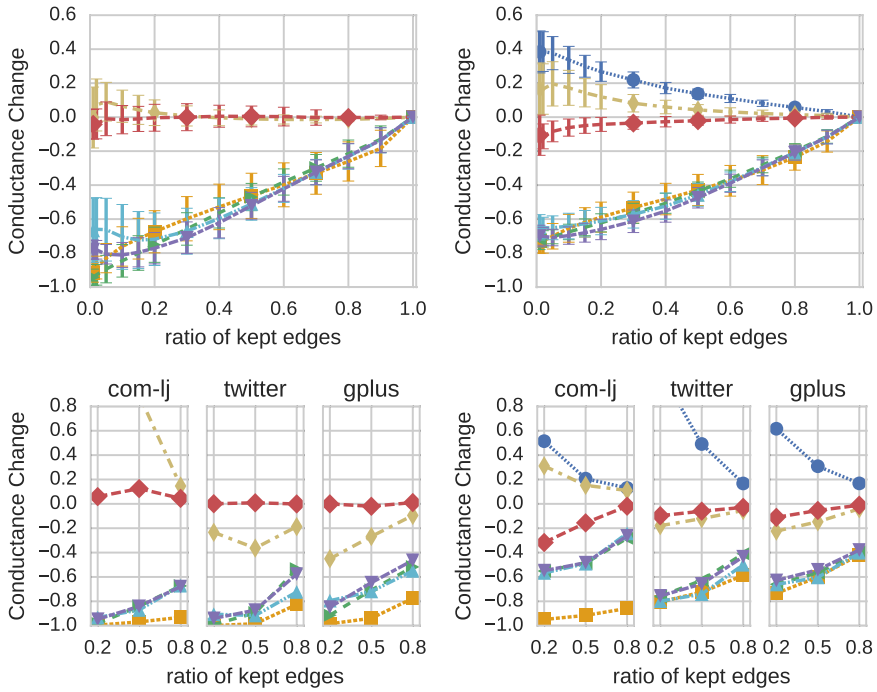
The similarity of the curves in Fig. 62 catches the eye immediately: For these node centrality measures, the sparsification methods behave in a very similar way. This similarity could be explained by strong correlations between node degree, PageRank and betweenness, which have been observed before (e.g. [FBFM08]). Note that betweenness centrality is also not exactly preserved on the original network; this is due to the approximation, which adds additional noise.

Random edge deletion and Local Degree perform best on most networks. In accordance with our intuition that edges leading to high-degree neighbors are important and should be preserved, our experiments show that the Local Degree method preserves all three considered node centralities. Nevertheless, random edge filtering with the additional local filtering step outperforms it concerning the preservation of Betweenness Centrality. The differences are small though and similar to those that are due to the approximation error so they might actually be caused by the approximation method for Betweenness centrality that behaves differently depending on the structure of the network. On the Facebook networks, Edge Forest Fire fails early while on the other networks it is among the best methods. As the expected number of randomly selected incident edges via the "burning process" of Edge Forest Fire is relatively low even for high-degree nodes, it fails at preserving node degrees. Nevertheless in the non-Facebook networks it seems to preserve enough important connections in order to preserve PageRank and betweenness centrali-

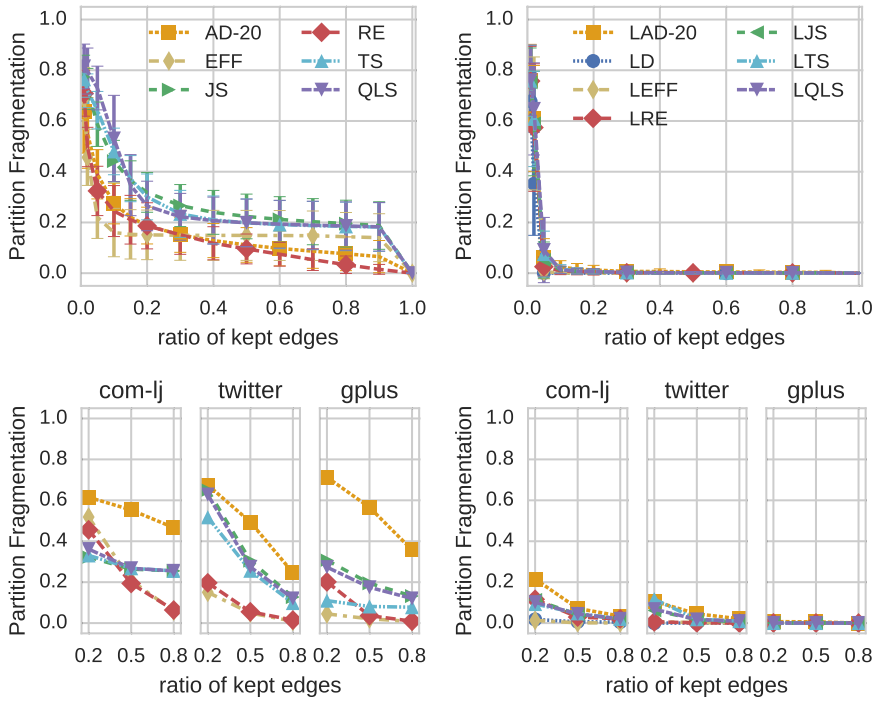
ties relatively well. Methods that are focused on keeping edges within dense regions are not as good at preserving these centralities. Adding the additional local filtering step again leads to a better preservation of the properties but does not change the general picture.

Community Structure

In order to understand how the community structure of the networks is maintained, we consider for each network a fixed partition into communities that has been found by the PLM algorithm (Section 8.2.2) on the original network. We report some properties of this community structure for each level of sparsification. There are many ways to characterize a community structure. We pick two properties of communities that we consider to be crucial. Communities are commonly described to be internally dense and externally sparse subgraphs. A natural measure is thus *conductance* (see Def. 25) which compares the size of the cut of a community to the volume of the community, i.e. the sum of all degrees (or the volume of the rest of the network if it should be larger). Low conductance values indicate clearly separable communities. We consider the average conductance of all communities. Furthermore we expect that communities are connected. In order to measure this, we introduce the fraction of the nodes in a community that does not belong to the largest connected component of the community as partition fragmentation. We report the average fragmentation of all communities. Results are shown in Fig. 63.



(d) Relative conductance change of a fixed partition



(e) Average partition fragmentation

Figure 63: Preservation of community structure

We plot the relative inter-cluster conductance change in Figure 63d. A value of 0 means that the conductance stays the same, a value of -1 indicates that the conductance became 0 (i.e. a decrease by 100%) and a value of 1 indicates that the conductance has been

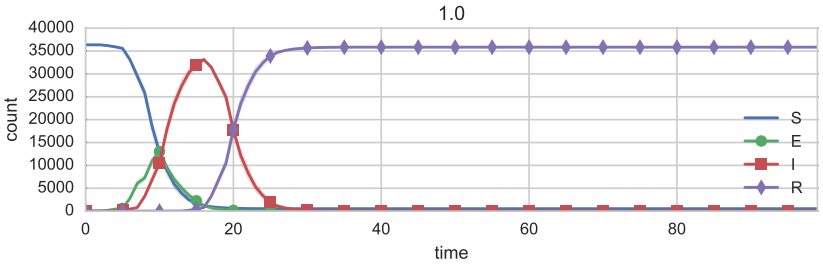
doubled (i.e. an increase of 100%). We can again see that there are three categories of algorithms: the first group consisting of random edge sampling preserves the conductance values on most networks. The second group contains only Local Degree and increases the conductance. Edge Forest Fire has no clear behavior.

The third group consisting of Jaccard Similarity, Simmelian Backbones and algebraic distance strongly decreases the conductance. With the additional local filtering step the decrease in conductance is not as strong but still very significant. The keeping of inter-community edges of the Local Degree method, which also explains why it preserves the connectivity so well, can be explained as follows: Consider a hub node x within a community with neighbors that are for the most part also connected to a hub node y with higher degree than x. Due to the way Local Degree scores edges, x will lose many of its connections within the community and may be pulled into the community of a neighboring high-degree node z that is not part of the original community of x. Jaccard Similarity, Simmelian Backbones and algebraic distance on the other hand focus – by design – on intra-community edges. Random edge sampling and Edge Forest Fire filter both types of edges almost equally distributed which is not surprising given their random nature. Depending on the network Edge Forest Fire shows different behavior, this indicates that these networks have a different structure.

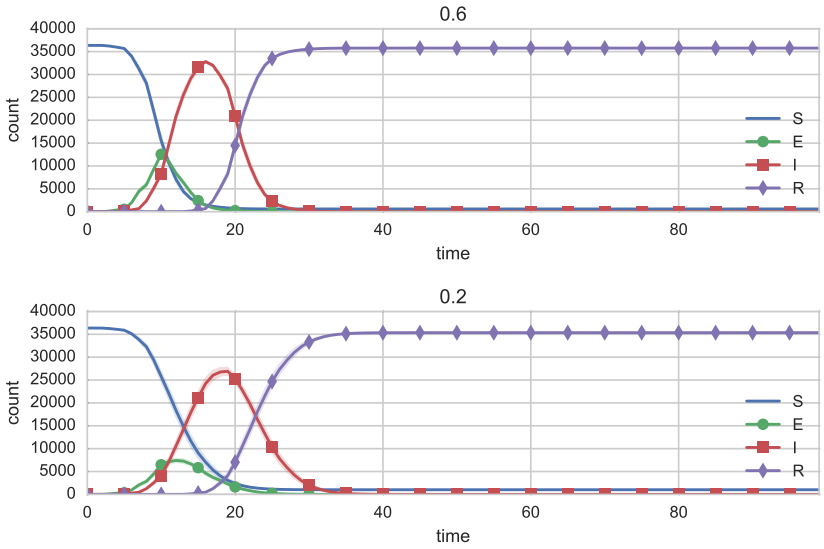
In Figure 63e it becomes obvious that only local filtering allows methods to keep the intra-cluster connectivity up to very sparse graphs. On the Facebook networks Simmelian Backbones and Jaccard Similarity without local filtering are actually the worst in this respect, they do not keep the connectivity even though they prefer intra-cluster edges as we have seen before. On the other networks, algebraic distance is even more extreme in this regard. Random edge sampling and Edge Forest Fire on the non-Facebook networks are the only non-local method where a slow increase of the fragmentation can be observed, all other methods lead to a steep increase of the fragmentation during the first 10% of edges that are removed.

11.4.3 *Epidemic Simulations*

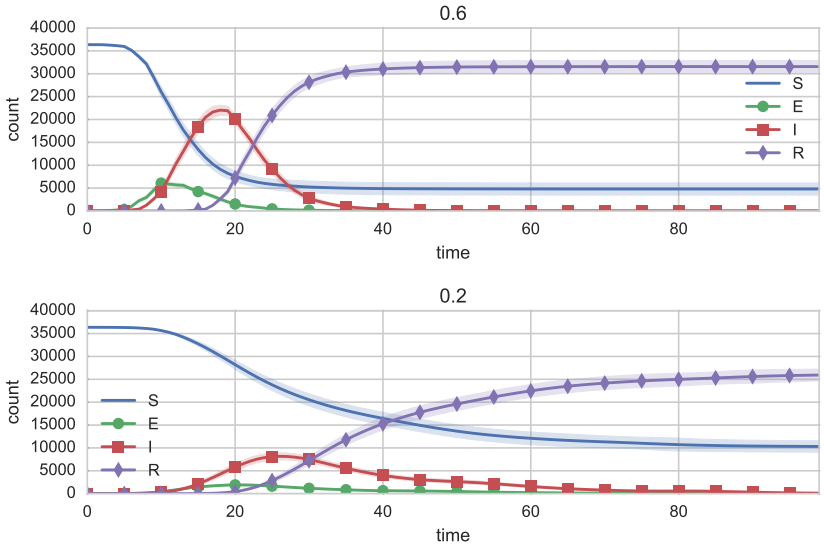
The previous experiments focused on static structural properties only. Now we briefly turn to dynamic, emergent properties that can be observed by simulating processes on networks. We employ the SEIR epidemic model as described in Sec. 2.6, and count nodes of the different states (susceptible (S), exposed (E), infectious (I), removed (R)) over time. We can ask whether sparsified versions of a network give rise to similar epidemiological dynamics, in terms of the size and timing of a disease outbreak, and add another level of analysis for the sparsification methods. We select `fb-Texas84` (ca. 1.6 million edges) as a representative social network and run the SEIR simulation 50 times with a latency period of 2 time steps, an infectious period of 9 time steps, and a transmission probability of 0.1. Figure 64a shows the aggregated epidemic curves (where the central line represents the median and the shaded areas around it the standard deviation) for the original network. While epidemic dynamics can depend strongly on the specific network structure, the following observations were roughly consistent across the Facebook-type networks.



(a) Epidemic curves of SEIR simulation on a Facebook social network



(b) Epidemic curves after Local Degree sparsification (ratios 0.6 and 0.2)



(c) Epidemic curves after algebraic distance sparsification (ratios 0.6 and 0.2)

Figure 64: Epidemic simulations

The Local Degree method most closely replicates the epidemic curves of the original down to an edge ratio of 0.2, producing only a minor delay in the outbreak and slightly

lower peak number of infected nodes, but an identical converged state (Fig. 64b). One reason for this is certainly that connectedness and short paths are preserved. It may also point to the importance of local hubs in epidemic propagation. Random edge sampling and Forest Fire sparsification also perform well and produce a similar high fidelity. Other methods deviate more from the original epidemic dynamics by delaying and dampening the outbreak, with some also strongly reducing the final number of infections. For Local Jaccard Similarity, thinning the network slows the outbreak slightly, leading to a less sharp and high peak of infected nodes, but reproduces essentially the same epidemic curve shapes and final state. At the other end of the spectrum, sparsification by algebraic distance (Fig. 64c) and the Simmelian methods selectively removes bottleneck edges between dense regions of the network. These edges are likely to be critically important for the propagation of a disease, and hence epidemic dynamics are significantly altered with deleting those edges.

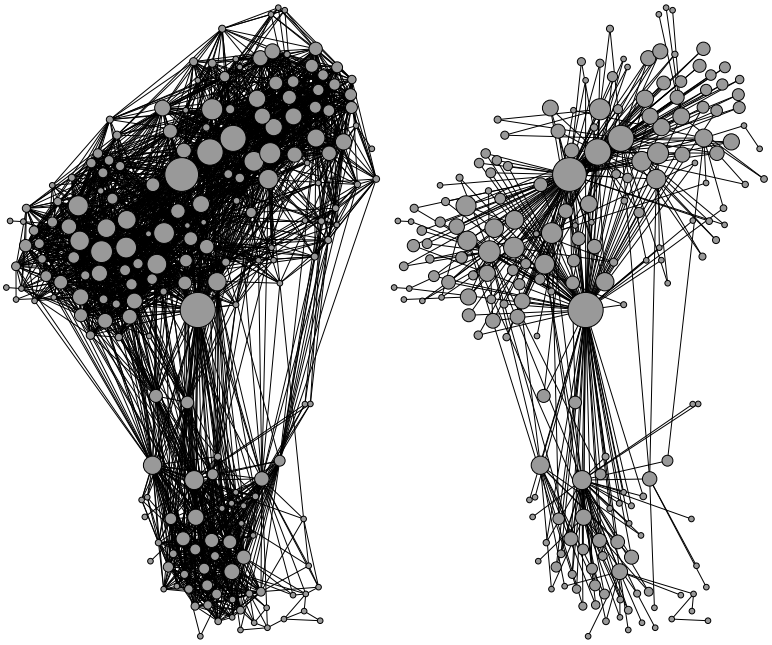


Figure 65: Drawing of the Jazz musicians collaboration network and the Local Degree sparsified version containing 15 % of edges. Node size proportional to degree.

CONCLUSION OF PART IV

Part IV focused on edge sparsification of networks, i.e. reducing the number of edges significantly while preserving essential properties of the original network. This is possible because not all edges contribute equally to important structural features. We show how methods that we and others have proposed can be implemented in a unified conceptual framework of computing edge centrality scores and filtering edges locally or globally by these scores. We hope that the conceptual framework of edge scoring and filtering as well as our evaluation methods are steps towards a more unified perspective on a variety of related methods that have been proposed in different contexts.

Our experimental study on networks from Facebook, Twitter and Google+ as well as synthetic networks shows that several sparsification methods are capable of preserving a set of relevant properties of social networks when up to 80% of edges have been removed. Random edge deletion performs surprisingly well and retains a wide range of properties, but more targeted methods can perform even better. We propose local filtering as a generally applicable and computationally cheap post-processing step for edge sparsification methods that improves the preservation of almost all properties as it leads to a more equal rate of filtering across the network. Simmelian Backbones, Jaccard Similarity and Algebraic Distance prefer intra-cluster edges and thus do not keep global structures but with the added local filtering step they are able to enforce and retain a community structure as it was already shown for Jaccard Similarity. Our novel method Local Degree, which is based on the notion that connections to hubs are highly important for the network's structure, in contrast preserves shortest paths and the overall connectivity of the network. This can be seen at the almost perfectly preserved diameter and the well-preserved behavior of the network in epidemic simulations. Depending on the network, the Local Degree method is also able to preserve clustering coefficients and centralities. Our adaption of the Forest Fire sampling algorithm to edge scoring depends strongly on the specific network's structure. It is good at preserving connectivity, on some networks it also preserves centralities and the diameter. Our results illuminate which methods are suitable with respect to which properties of a network. Additionally, we take a look at emergent properties by simulating epidemic spreading on sparsified networks in comparison with the original network. We conclude that our Local Degree method is best for preserving connectivity and short distances which results in a good preservation of the diameter of the network, some centrality measures and the behavior of epidemic spreading.

Furthermore, we have published efficient parallelized implementations and a framework for such methods as part of NetworKit (see Part II). While our study covers various approaches from the literature, it is by no means exhaustive due to the vast amount of potential sparsification techniques. With future methods in mind, we hope to contribute a framework for their implementation and evaluation.

Part V

GENERATIVE MODELS FOR REALISTIC SYNTHETIC NETWORKS

12 INTRODUCTION TO GENERATIVE NETWORK MODELS

> In the days when Sussman was a novice, Minsky once came to him as he sat hacking at the PDP-6.
>
> "What are you doing?", asked Minsky. "I am training a randomly wired neural net to play Tic-tac-toe", Sussman replied. "Why is the net wired randomly?", asked Minsky. "I do not want it to have any preconceptions of how to play", Sussman said.
>
> Minsky then shut his eyes. "Why do you close your eyes?" Sussman asked his teacher. "So that the room will be empty." At that moment, Sussman was enlightened.
>
> – traditional hacker koan

This chapter provides a brief introduction to the topic of generative network models with the aim of creating synthetic graphs that match important properties observed in real complex networks. It presents relevant related work leading up to Chapter 13. Aleksejs Sazonovs helped with initial literature research.

12.1 APPLICATIONS OF GENERATIVE NETWORK MODELS

Generative models play a central role in the emerging field of network science. After observing certain statistical patterns in networks, we ask whether structurally similar graphs can be generated by following simple formal rules. With the term generative network model, we refer to an algorithm which given some input parameters produces a network according to a well-defined procedure.

On the one hand, software and algorithm developers want generators for synthetic datasets which can be scaled and parametrized and produce graphs which resemble the real application data. On the other hand, network scientists need models to increase their understanding of network phenomena: What are the rules that guide the formation and evolution of the complex networks we observe? If we observe common structures in very different domains (e.g. as distinct as society and biochemistry), is it because common abstract principles are at work? What are "typical" and "untypical" structures that emerge?

Main use cases for generative network models include:

Obfuscation: In scenarios where the actual network data is not to be disclosed (out of privacy concerns, commercial interest etc.), accurate replicas can be used for obfuscation.

Extrapolation and Sampling: If a generative model can be fitted to a real network and recreate important properties, it can be used to produce realistic synthetic data sets at different scales. Larger graphs allow for testing the scalability of algorithms or, assuming network growth over time, simulation of the network's future development. Likewise, smaller replicas are useful for algorithm testing.

Compression: Graph compression is a another possible application for generative models which are able to realistically replicate the features of a given network. By storing only the parameters of the model, significant compression can be achieved. However, the models we consider are suitable for replicating statistical properties of the data rather than the original graph itself.

12.2 EXEMPLARY MODELS

A comprehensive overview of the multitude of existing generative network models would go far beyond the scope of this thesis, hence we refer the reader to surveys (e.g. Goldenberg et al. [GZFA10]) for a bird's eye view of the topic. The selection of models that follow is certainly to some extent arbitrary, having in common that we have encountered the use of these models in the graph algorithms research community, and for most of them efficient implementations in NetworKit are at hand. However, an underlying idea is to cover both simple, fundamental models, like Erdős–Rényi random graphs, and more complex models that come with some claim of comprehensive realism across multiple structural properties of real networks. We also select models for which there are generative algorithms that are similar to NetworKit's analysis algorithms in computational efficiency, so that they are capable of producing millions of edges in a relatively short time.

The *Erdős–Rényi model* $ER(n, p)$ is a simple probabilistic model, which creates edges among n nodes with a uniform probability of p for each of the $\{u, v\}$ pairs [P. 60]. Not intended to generate realistic networks, it has nonetheless been a very successful spark for graph theory research, and properties of the model have been subject to much theoretical analysis. For example, it can be shown that a graph generated by $ER(n, p)$ is almost certainly connected if $p \geq 2 \cdot \ln(n) \cdot (1/n)$.

The degree distribution of an Erdős–Rényi graph resembles a normal distribution, unlike the highly skewed degree distribution often found in real-world networks, where few high-degree nodes stand vis-à-vis a large number of low-degree nodes. This motivated the *Barabasi-Albert model* $BA(n, k)$ [AB02], which implements a process in which new nodes preferentially attach themselves to existing high-degree nodes. The model adds nodes one by one, with k edges attached to a new node. The probability that such an edge will attach to an existing node v is $p(v) = \frac{deg(v)}{\sum_{u \in V} deg(u)}$. This process has been shown to result in a power-law degree distribution that is a common feature of real complex networks. Other frequently found properties are not targeted, such as the closure of triangles (clustering) or a modular composition out of dense subgraphs (communities).

Another family of generative models has the aim of replicating any given degree distribution. The Chung-Lu (CL) model [ACL00] is a random graph model which aims to replicate a given degree distribution. Given a degree sequence, the method creates edges between nodes with a probability of $p(u, v) = \frac{deg(u)deg(v)}{\sum_k deg(k)}$ and recreates the degree sequence in expectation.

For a given realizable degree sequence, the algorithm of *Havel* [Hav55] and *Hakimi* [Hak62] generates a graph with exactly this degree sequence. Unlike the Chung-Lu model, the generative process promotes the formation of closed triangles, leading to a higher clustering coefficient. Since the Havel-Hakimi algorithm is deterministic but our focus is on generative models that incorporate randomness, it is more appropriate to compare with the *Edge-Switching Markov Chain Generator* (see e.g. [MKI+03]), which uses the Havel-Hakimi algorithm initially but modifies the graph through random edge swaps. This process converges in a graph that is drawn uniformly at random from all possible graphs with the given degree sequence. Even though no theoretical bound is known for the number of needed swaps, experiments have shown that between $10 \cdot m$ and $100 \cdot m$ swaps should be enough in practice [MKI+03], we decided to use $10 \cdot m$ in order to not to further increase the already large running time. While there are computationally cheaper

algorithms that try to generate a similar graph, they actually generate significantly different graphs, see also [SHZ15] for a more in-depth discussion of these different generative models.

While the previous models target only the degree distribution, the next group of models are aimed at creating synthetic networks that are comprehensively realistic with respect to several aspects.

The Recursive Matrix (RMAT) model [CZF04] was proposed to recreate various properties of complex networks, including an optional power-law degree distribution, the small-world property and self-similarity. Design goals also include few parameters and high generation speed. It is classified by the authors as a procedural model, i.e. event-driven, and therefore easily enables dynamic graph generation. Three goals for the model are singled out and emphasized: Generated graphs should a) match a given degree distribution (power-law or other) b) exhibit community structure and c) have a small diameter.

RMAT operates on the initially empty adjacency matrix A of the result graph, an $n \times n$ matrix where n is a power of two. The matrix is recursively subdivided into four equal-sized quadrants. The quadrants are assigned the probabilities a, b, c, d which add up to 1. Edges (i.e. 1-entries) are "dropped" into the matrix and land in one of the quadrants according to their probabilities. Then the process continues recursively by further subdividing the quadrant into four parts with probabilities a, b, c, d, until a 1×1 partition is reached, which determines the location of the edge. RMAT natively generates directed graphs, undirected graphs are generated by treating the edges as undirected and removing duplicate edges.

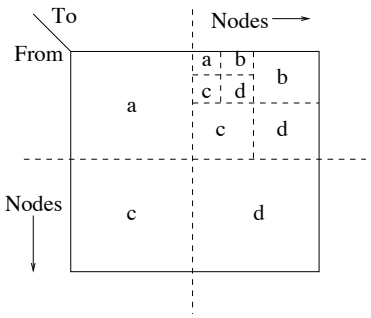


Figure 66: R-MAT model: Subdivision of the adjacency matrix (Source: [CZF04])

According to the authors, RMAT intuitively generates communities in the graph. Quadrants a and d define two groups of nodes, while b and c contain the edges between those groups. Assuming that $a, d \geq b, c$, the groups are internally dense and externally sparse, forming communities. The recursive procedure has been claimed to generate a hierarchical community structure by forming sub-communities. However, the generative mechanism does not promote the formation of closed triangles, which can be considered the building blocks of community structure.

LFR is a method often used to generate benchmark graphs for community detection algorithms [LFR08]. The generated graphs allow a distinct community structure and "ground truth" information about the communities each node belongs to. A mixing parameter μ determines the number of inter- and intra-community edges for a given node u: $\mu \cdot \deg(u)$ inter-community and $(1 - \mu) \cdot \deg(u)$ intra-community edges will be created, enabling the generation of community detection test instances with varying level

of difficulty (cf. Sec. 7.3). Furthermore, LFR generates a power-law distribution of node degrees and community sizes. LFR is the basis for our new generative algorithm LFR+, and is therefore described in detail in Sec. 13.4.

BTER [KPPS13] is a two-stage structure-driven model, which combines aspects of the ER and CL model. It uses the standard ER model to form relatively dense subgraphs, thus forming distinct communities. Afterwards, the CL model is used to add edges, allowing to match the expected degree distribution [SKP11].

Some models use geometric methods to generate graphs. The *Hyperbolic Unit-Disk Graph* model (also known as *Hyperbolic Random Graph* model) embeds nodes into hyperbolic instead of Euclidean geometry and connects them if they are within a certain distance from each other [KPK+10]. Analysis shows that due to the properties of hyperbolic space the model is able to replicate some properties observed in real networks, such as a power-law degree distribution. A new parallel implementation in NetworKit reduces the computation cost to subquadratic time [vLMP15].

Editing models are a special class of generative models which create a synthetic network by editing a full initial network. The resulting replica network should both preserve structural properties of the original and introduce variety. Editing models are geared towards scenarios where synthetic data sets similar to real datasets are required, e.g. for anonymizing networks, generating test data for algorithm evaluation, avoid disclosure of the actual data or generating plausible hypothetic scenarios.

MUSKETEER *(Multiscale Entropic Network Generator)* [GSM15] is an editing model using a multi-level (or multi-scale) approach. Inspired by the algebraic multigrid scheme for solving linear algebra problems, the initial graph is repeatedly coarsened and then interpolated back to the original scale. At each level of the coarsening/refinement process, edits to the graph are performed. Primitive edits on a coarse level, e.g. the insertion of a node, result in larger-scale changes on the fine level, e.g. the insertion of a cluster of nodes. Through an evaluation starting with an empirical network, it is shown that on average structural properties such as number of nodes, number of edges, number of components, clustering coefficient, average degree, degree assortativity, betweenness centrality and modularity are preserved with high accuracy. The MUSKETEER model thereby matches structural properties more closely than other models. Running time is linear in the number of edges. An implementation is freely available [AG].

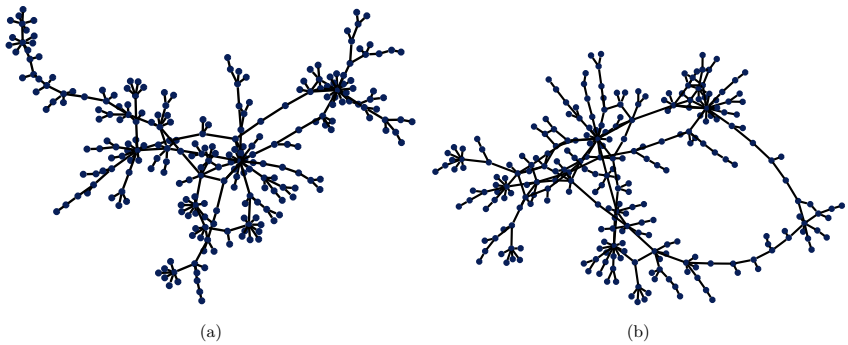


Figure 67: A network used for epidemiological research (a) and its replica produced by MUSKETEER (b). Source: [GSM15]

13

GENERATING SCALED REPLICAS OF REAL-WORLD NETWORKS

> Early one morning, a programmer asked the great master: "I am ready to write some unit tests. What code coverage should I aim for?" The great master replied: "Don't worry about coverage, just write some good tests." The programmer smiled, bowed, and left.
>
> Later that day, a second programmer asked the same question. The great master pointed at a pot of boiling water and said: "How many grains of rice should put in that pot?" The programmer, looking puzzled, replied: "How can I possibly tell you? It depends on how many people you need to feed, how hungry they are, what other food you are serving, how much rice you have available, and so on." "Exactly," said the great master. The second programmer smiled, bowed, and left.
>
> Toward the end of the day, a third programmer came and asked the same question about code coverage. "Eighty percent and no less!" Replied the master in a stern voice, pounding his fist on the table. The third programmer smiled, bowed, and left.
>
> After this last reply, a young apprentice approached the great master: "Great master, today I overheard you answer the same question about code coverage with three different answers. Why?" The great master stood up from his chair: "Come get some fresh tea with me and let's talk."
>
> After they filled their cups with steaming hot green tea, the great master began to answer: "The first programmer is new and just getting started with testing. Right now he has a lot of code and no tests. He has a long way to go; focusing on code coverage at this time would be depressing and quite useless. He's better off just getting used to writing and running some tests. He can worry about coverage later. The second programmer, on the other hand, is quite experienced both at programming and testing. When I replied by asking her how many grains of rice I should put in a pot, I helped her realize that the amount of testing necessary depends on a number of factors, and she knows those factors better than I do – it's her code after all. There is no single, simple, answer, and she's smart enough to handle the truth and work with that."
>
> "I see," said the young apprentice, "but if there is no single simple answer, then why did you answer the third programmer 'Eighty percent and no less'?" The great master laughed... "The third programmer wants only simple answers – even when there are no simple answers ... and then does not follow them anyway."
>
> The young apprentice and the grizzled great master finished drinking their tea in contemplative silence.
>
> – a koan by Beth Anderson

In the following, we present the first extensive study on the ability of generative models to produce realistic scaled replicas of original networks. Large synthetic graphs that match important properties of real networks are important for engineering computational methods on networks, since large real networks can be scarce or otherwise inaccessible in many scenarios. We fit the parameters of existing models to original networks in an attempt to replicate their properties as closely as possible, and furthermore parametrize the models so that they produce scaled-up replicas, containing a multiple of the originals nodes. We introduce the LFR+ generator, a more flexible modification of the LFR model used for community detection benchmarking, including an implementation for both models in NetworKit that is faster than the reference implementation. Most importantly, we

show show that in comparison with other relevant generative models LFR+ produces overall the most realistic replicas according to a wide range of criteria.

This chapter is based on joint work with Ilya Safro, Alexander Gutfraind, Henning Meyerhenke and Michael Hamann, which is recent and so far unpublished.

13.1 CONTEXT AND CONTRIBUTION

When engineering algorithms, the ability to create good synthetic test data sets is a valuable tool to estimate effectiveness and scalability of proposed methods. In the context of developing algorithms for complex network analysis problems, we often need to deal with a lack of realistic large-scale graphs, since real application data is often proprietary, sensitive, or otherwise unavailable. Realistic synthetic graphs allows us to generate experimental results which are representative for what can be observed for real data. Realism has frequently been one of the primary aspects under which generative network models have been studied. Work by Leskovec et al. [LF07] addresses the problem of generating a synthetic graph that matches the properties of a given large real network, going as far as to conclude that their replicas can serve as "anonymized" (i.e. obfuscated) substitutes when the real network cannot be shared. Their approach, based on the widely used RMAT generator, is described and evaluated in Sec. 13.5.3. Other current models that have been presented with a claim of comprehensive realism, as well as high scalability, are the BTER model [KPPS13] and the Hyperbolic Unit Disk Graph in its most current implementation by von Looz et al. [vLMP15]

We clarify what constitutes realism for the replica, and conceptualize it in different ways, as matching an original graph in a set of important structural properties, and as matching the running times of various graph algorithms (i.e., we compare the distribution of running times on sets of originals and replicas). We specifically pursue the goal of creating realistic scaled replicas of complex networks. Starting from an initial real network, we want to create a synthetic graph that is larger by orders of magnitude but preserves important aspects of realism. Based on the mechanisms of the LFR generative model, we design the LFR+ generator and a model fitting scheme and show that it is superior in terms of realism. The resulting implementation is scalable, capable of creating realistic scaled replicas on the scale of 10^8 edges in minutes.

13.2 PROBLEM DEFINITION AND DESIGN GOALS

We envision two usage scenarios for the methods studied and developed in the following: Given a real network O (having n_o nodes) that cannot be freely shared, we would like to be able to create synthetic network R (with n_r nodes) that matches the original in essential structural properties, so that computational results obtained from processing this network are representative for what the original network would yield. We refer to R as a *replica*. We assume that whoever creates the replica has access to the original and can pass it to a *model fitting* algorithm which uses it to parametrize a generative model. Moreover, in addition to producing *scale-1 replicas* (where $n_r = n_o$), we want to use the generative model for *extrapolation*: We want to parametrize it so that it produces a *scaled replica* R^x that has $n_r = x \cdot n$ nodes, where x is called the *scaling factor*. The structural properties of R^x should be such that they resemble a later growth stage of the original (a criterion discussed in more detail in Sec. 13.2.1). This should enable users

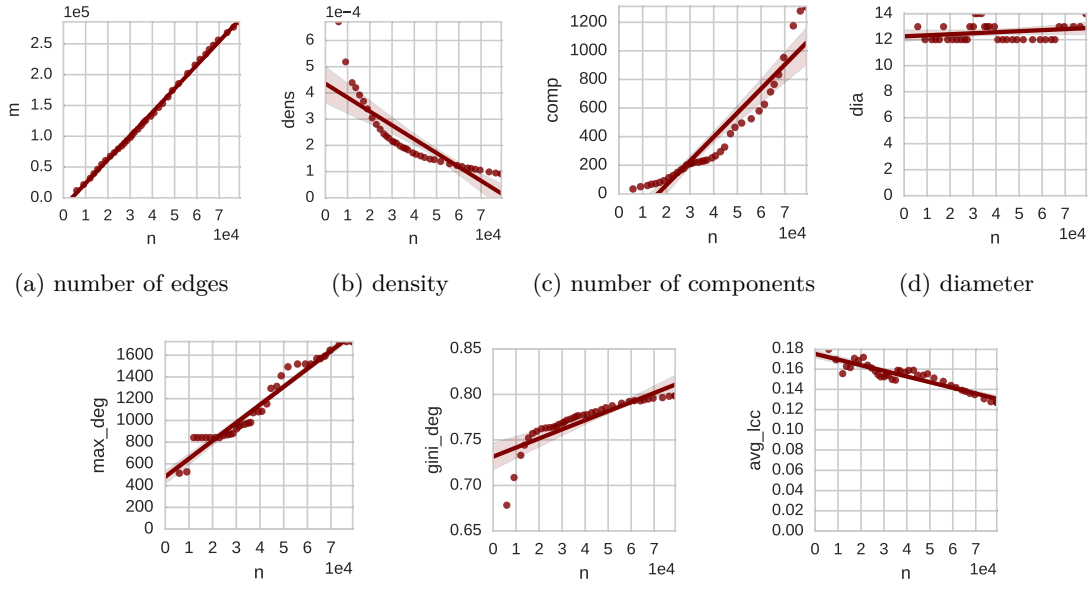
of the replica to extrapolate the behavior of their methods when the network data is significantly scaled.

With respect to performance, we would like the generator algorithm and implementation to be efficient enough to produce large data sets (on the order of several millions of nodes and edges) in practically short time. Furthermore, we also require the fitting scheme to be efficient.

Why is realism important? If synthetic graphs are used for algorithmic experiments, we need to keep in mind that the performance of graph algorithms can be highly structure-dependent. As pointed out in [Joh02], random, synthetic instances are very useful in experimental algorithmics, but these advantages "can be substantially dissipated if there is no evidence that the random instance class says anything about what will happen for real-world instances". Generating and examining this kind of evidence is therefore a central theme here. We consider a generative model realistic if there is high structural similarity between the synthetic graphs produced and relevant real-world networks that serve as input in a given application context. "Structural similarity" is understood as a gradual and statistical concept, as discussed before in the context of sparsification (Ch. 11). We do not aim for an exact correspondence between original and replica, with the extreme case of isomorphic graphs, since this is in general not a requirement for generalizable experiments. In fact, producing some structural variance is a desired feature of an effective generator. Structural similarity is quantified using a collection of measures for characterizing network structure (cf. Chapter 2), but this includes necessarily a selection of properties that are considered essential.

13.2.1 *Scaling Behavior of Real-World Networks*

What is a realistic *scaled* replica of a network? There are at least two subtly different lines of attack when attempting a definition for this. We can understand realistic scaling as either a) recreating the scaling behavior observed in real-world networks, or b) preserving a given set of properties of the original as closely as possible, given that the number of nodes and edges grows. An exploratory look at the scaling behavior of real-world networks is provided in Figures 68 and 69. A set of basic structural measures is plotted against the number of nodes n, as well as a regression line and confidence intervals (shaded area) to emphasize the trend. Figure 68 shows the evolution of a dynamic network compiled from the Enron email corpus, a set of corporate email communication records that became public in the wake of the Enron accounting fraud scandal [DFC05]. For this experiment, the raw data set was mined with a parser script to construct a network as follows: Nodes represent email addresses and an undirected, unweighted edge is added if at least one email has been sent. Dynamic network data is represented as a stream of graph modifier events (add node, add edge, ...) and we look at snapshots of the graph constructed this way at every 10^5 events. What can be observed from this is a growth of m that is linear in n, falling density (which follows mathematically from linear edge growth), an essentially constant small diameter of the largest connected component, a growing maximum node degree, an increase in the skewedness of the node degree distribution as measured by the Gini coefficient [Gin12], and a slightly falling average local clustering coefficient. Figure 69 reports these measures not for the actual temporal evolution of a single network, but for a set of 100 Facebook social networks (already introduced in Ch. 10). They were collected at an early stage of the Facebook online social networking service, in which



(a) number of edges (b) density (c) number of components (d) diameter

(e) maximum degree (f) Gini coefficient of degree distribution (g) average local clustering coefficient

Figure 68: Scaling behavior of Enron email network. Data points and regression model shown.

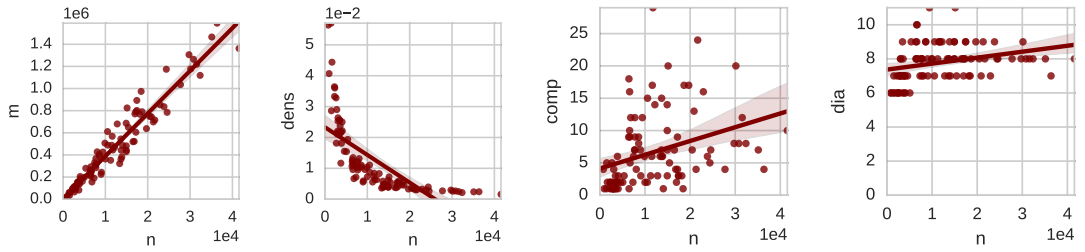
networks were still separated by universities. Since these networks were formed by the local actions of a collection of social actors which have essentially similar behavior, the mechanism of growth by which the networks have assembled themselves is essentially the same. These networks of different sizes can in a sense be treated as differently scaled versions of "the" Facebook social network. The observed trends generally agree with those of the Enron communication network.

Attempts to formulate universal scaling laws for all complex networks should be viewed critically, since scaling behavior is a property of the system for which we apply a network model and may vary widely. Nonetheless, the trends represented here are commonly observed, and we use them to define desired scaling properties for the remainder of the study as follows:

- m grows linearly with n
- the diameter remains essentially constant, preserving the "small world property"
- the shape of the degree distribution remains skewed
- the maximum node degree increases
- the number of connected components may grow, but a giant connected component containing the majority of nodes is maintained

13.3 ABANDONED APPROACH: A MULTISCALE GENERATOR FOR SCALED REPLICAS

We initially considered a combination of algorithmic methods used for MUSKETEER (Multiscale Entropic Network Generator) [GSM15] as a candidate approach for the goal



(a) number of edges (b) density (c) number of components (d) diameter

(e) maximum degree (f) Gini coefficient of degree distribution (g) average local clustering coefficient

Figure 69: Scaling behavior of 100 Facebook networks

of creating realistic scaled replicas. As described in Sec. 12.2, MUSKETEER implements an editing model that coarsens a given input graph and edits it at various levels of the coarsening hierarchy through node and edge insertions and deletions. Copying coarse nodes, and thereby copying entire subgraphs, can be used to scale up the network. The multiscale method's ability to simultaneously reproduce substructures of different scales was thought to be a promising approach for achieving realism. In fact, MUSKETEER has been previously used to realistically replicate an original with an increase in size, albeit by very small scaling factors (e.g. $x = 2$). The restricted scaling factor is not incidental: After copying a node on a coarse level of the coarsening hierarchy and connecting it with a new coarse edge, MUSKETEER uncoarsens this edge into a random graph. For larger scaling factors, a large part of edges will be generated this way, causing the synthetic network to degenerate into a random graph and losing realism. Our attempts to modify the algorithm could not satisfactorily solve the problem of connecting a large number of coarse node copies (i.e. copies of subgraphs of the original) in a realistic way. Moreover, considering a large number of possibly parameter-dependent heuristics (including aggregation schemes, e.g. by matching with edges weighted by different edge centrality measures, building of the coarse graph hierarchy, heuristics for copying and reconnecting coarse nodes, node and edge editing schemes etc.) would have complicated algorithm engineering, as well as leading to a rather complex implementation which is difficult to optimize. While this does not disprove that multiscale editing is a potentially viable approach, the unsolved algorithmic questions and the inherent complexity of the method prompted us to abandon it in favor of the simpler and more promising LFR+ model.

In preliminary experiments, the original MUSKETEER performed well in terms of realism. However, we do not include it in the following experimental study for two reasons: As the only editing model, MUSKETEER has an "unfair advantage" for scale-1 repli-

cas, because with low editing rates significant amounts of the original graph are copied directly into the replica. Secondly, the available Python implementation is significantly slower than all other implementations in the study, and not practical for the million-edge graphs contained in our test sets.

13.4 THE LFR+ GENERATOR

We introduce LFR+, a graph generator based on the LFR generative model [LFR08], with an efficient implementation in NetworKit. Compared to its predecessor, LFR+ allows for better fitting to input networks and achieves increased realism, as shown in the following. We first review the original LFR model, then describe changes to the algorithm that yield LFR+.

13.4.1 *Original LFR Model*

The LFR+ generator is a modification of the LFR generator proposed by Lancichinetti, Fortunato and Radicchi for creating benchmark graphs for testing community detection algorithms (cf. Sec. 7.3) [LFR08]. The authors originally introduced LFR as an improvement over previous generators used for community detection benchmarks, which employed a simple block model in which edges were created uniformly at random according to a higher and a lower probability depending on whether the node pair belongs to the same or different subsets of a given partition (e.g. [GN02] [GKS+12]). This fails to generate the skewed degree distribution often encountered in real complex networks. The "vanilla" LFR generator assumes that both the degree and the community size distributions are power laws, with their exponents γ and β being passed as parameters. The graph structure is strongly influenced by the choice of a *mixing parameter* μ: Each node shares a fraction $1 - \mu$ of its incident edges with the other nodes of its community and a fraction μ with the other nodes of the network. The smaller μ is chosen, the fewer inter-community edges exist and the more distinctive is the community structure.

The algorithm for the generation of the LFR model in [LFR08] differs from the one described in [LF09b]. The latter LFR algorithm, which is also used in the implementation provided by the authors, can be briefly summarized as follows:

1. each node u is given a degree $\deg(u)$ sampled from a power-law distribution, determined by an exponent γ as well as an average and maximum degree
2. community sizes are sampled from a power-law distribution with exponent β and given minimum and maximum community size
3. nodes u are assigned randomly and iteratively to communities so that their internal degree $(1 - \mu) \cdot \deg(u)$ can be satisfied by the size of the community
4. the Edge-Switching Markov Chain Generator (also called fixed-degree sequence model, see e.g. [MKI+03]) is used to generate a graph per community using the internal degrees of the nodes and a global graph using the remaining degrees of the nodes
5. a rewiring step switches edges of the global graph such that no additional intra-community edges are introduced by the global graph

In our implementation we parallelize the generation of the graphs per community as they are independent of each other.

13.4.2 *Modification into LFR+*

On a closer look, the generative mechanisms used in the LFR model allow for more flexibility: We do not need to restrict the distribution of node degrees to power-law distributions, but can produce arbitrary realizable degree sequences. The same is true for community sizes. This is a relevant extension, because neither are the degree distributions nor community size distributions encountered in real networks restricted to power-laws.

Our modification thus replaces the first three steps of the LFR generator. Instead of generating a degree distribution, we use the degrees of the original network. For the next steps, we detect a community structure of the original network and use this community structure instead of a random assignment. As we keep the node ids of the original network this also means that each community has still the nodes of the same degrees as in the original network. Instead of using a global value of μ, we assign a separate value per node as already mentioned as a possible extension in [LF09b].

In order to calculate the necessary sequence of μ values, we introduce the *local partition coverage* $\lambda_\zeta(u)$, which expresses how strongly a node is embedded into its community. A node u has the maximum $\lambda_\zeta(u)$ of 1.0 with respect to a partition ζ if it is connected only to nodes within its own community.

Definition 34 (Local Partition Coverage)**.** Given a graph $G = (V, E)$ and a partition ζ of V, *local partition coverage* is defined as

$$\lambda_\zeta(u) := \frac{|\{v \in N(u) : \zeta(v) = \zeta(u)\}|}{\deg(u)} \tag{34}$$

□

To let LFR+ replicate a given network $O = (V, E)$ we set the sequence $(\mu(u))_{(}u \in V)$ to $(1 - \lambda_\zeta(u))_{u \in V}$.

A nice property of LFR+ is that the resulting synthetic graph, in addition to replicating important properties with high fidelity, naturally produces random variance among the set of replicas.

13.5 FITTING GENERATIVE MODELS TO INPUT GRAPHS

A fitting scheme is an algorithm that takes a network as input and estimates parameters of a generative model. For this work, we test straightforward fitting schemes. We do not claim that they are the only possible or optimal schemes. Exploring different schemes would be relevant future work, but we restrict this study to one promising parametrization each.

Table 11 includes the model parameters we set, given an original network $O = (V, E)$ with $n = |V|$, $m = |E|$ and maximum degree $d_{\max}$. We denote as $(a_i)_{i \in M}$ a sequence of elements a with indexes i from an (ordered) set M, and we note $\cup_{j=0}^{k}(a_i)_{i \in M}$ the concatenation of k sequences.

We first discuss fitting of power-law distributions, which applies to several models, and then continue with a discussion of the parametrization of each model.

1 Or a higher value that is fitted for our minimum chosen power law exponent 1 using binary search such that the expected average community size is the actual average community size if the expected average community size would be too small otherwise.

model	parameters	fitting	fitting & scaling by $x \in \mathbb{N}$
Erdős–Rén	$ER(n', p)$	$n' = n$ $p = \frac{2m}{n \cdot (n-1)}$	$n' = x \cdot n$ $p = \frac{2m}{x \cdot n \cdot (n-1)}$
Barabasi-Albert	$BA(n', k)$	$n' = n$ $k = \lfloor m/n \rfloor$	$n' = x \cdot n$ $k = \lfloor m/n \rfloor$
Chung-Lu	$CL(d)$	$d = (\deg(u))_{u \in V}$	$d = \cup_{i=1}^{x} (\deg(u))_{u \in V}$
Edge-Switching Markov Chain	$EMC(d)$	$d = (\deg(u))_{u \in V}$	$d = \cup_{i=1}^{x} (\deg(u))_{u \in V}$
R-MAT	$RM(s, e, (a, b, c, d))$	$s = \lceil \log_2 n \rceil$ $e = \lfloor m/n \rfloor$ $(a, b, c, d) = \mathsf{kronfit}(O)$	$s = \lceil \log_2 x \cdot n \rceil$ $e = \lfloor m/n \rfloor$ $(a, b, c, d) = \mathsf{kronfit}(O)$
Hyperbolic Unit-Disk	$HUD(n, \bar{d}, \gamma)$	$n = n$ $\bar{d} = 2 \cdot (m/n)$ $\gamma = \mathsf{plfit}((\deg(u))_{u \in V})$	$n = x \cdot n$ $\bar{d} = 2 \cdot (m/n)$ $\gamma = \mathsf{plfit}((\deg(u))_{u \in V})$
BTER	$BTER(d, c)$	$d = (n_d)_{d \in (0, \ldots, d_{\max})}$ $c = (c_d)_{d \in (0, \ldots, d_{\max})}$	$d = (n_d \cdot x)_{d \in (0, \ldots, d_{\max})}$ $c = (c_d)_{d \in (0, \ldots, d_{\max})}$
LFR	$LFR(n', \gamma, \bar{d}, d_{\max}, \beta, c_{\min}, c_{\max})$	$n' = n$ $\gamma = \mathsf{plfit}((\deg(u))_{u \in V})$ $\bar{d} = 2 \cdot (m/n)$ $d_{\max} = \max((\deg(u))_{u \in V})$ $\zeta_s = \{\lvert C \rvert \mid C \in \mathsf{PLM}(O)\}$ $\beta = \mathsf{plfit}(\zeta_s)$ $c_{\min} = \min(\zeta_s)^1$ $c_{\max} = \max(\zeta_s)$	$n' = x \cdot n$ $\gamma = \mathsf{plfit}((\deg(u))_{u \in V})$ $\bar{d} = 2 \cdot (m/n)$ $d_{\max} = \max((\deg(u))_{u \in V})$ $\zeta_s = \{\lvert C \rvert \mid C \in \mathsf{PLM}(O)\}$ $\beta = \mathsf{plfit}(\zeta_s)$ $c_{\min} = \min(\zeta_s)^1$ $c_{\max} = \max(\zeta_s)$
LFR+	$LFR + (n', \zeta, \mu)$	$n' = n$ $\zeta = \mathsf{PLM}(O)$ $\mu = (1 - \lambda_\zeta(u))_{u \in V}$	$n' = x \cdot n$ $\zeta = \cup_{i=1}^{x} \mathsf{PLM}(O)$ $\mu = \cup_{i=1}^{x} (1 - \lambda_\zeta(u))_{u \in V}$

Table 11: Parameters set to fit a model to a given graph, and to produce a scaled-up replica

13.5.1 *On Fitting Degree Power Laws*

The LFR generator and the HUDG generator generate graphs with a power law degree distribution. Therefore at least the power law exponent, and, in the case of the LFR generator, also the average and maximum degree need to be determined such that the degree distribution fits the real network. In Table 11, plfit refers to our custom power law fitting scheme. There are many definitions of a fit, and in the case where the real distribution is not a power law distribution they can also lead to very different parameters. In general it is assumed that a power law distribution only holds starting with a minimum degree $x_{\min}$. The powerlaw module [ABP14] by default tries to find the best possible $x_{\min}$ which can lead to a very high minimum degree in the case of real networks where the degree distribution is not an exact power law distribution. While the minimum degree can be given explicitly, in our experience this also leads to higher degrees on average than in the real network. For the replication of a graph the average degree should be

kept as otherwise the graph will be much denser or sparser. At the same time, also the minimum and the maximum degree should be kept as otherwise the structure of the graph will be different as the presence of degree-1 nodes or very large hubs is a fundamental property. Therefore we fit the power law exponent such that with the given minimum and maximum degree the average degree of the real network is expected when a degree sequence is sampled from this power law distribution. Using binary search in the range of $[-6, -1]$ we repeatedly calculate the expected average degree until the power law exponent is accurate up to an error of 10^{-3}.

13.5.2 *Erdős–Rényi, Barabasi-Albert, Chung-Lu and ESMC*

The Erdős–Rényi model does not provide many options for parametrization. The edge probability p is set to produce the same edge-to-node ratio $\frac{m}{n}$ as the original. Likewise, we set the number of edges k coming with each new node in the Barabasi-Albert model to the edge-to-node ratio. The Chung-Lu and Edge-Switching Markov Chain generators simply receive the degree sequence of the original as input. In order to achieve scaling, x copies of this sequence are concatenated, multiplying the number of nodes by x while keeping the relative frequency of each degree.

13.5.3 *RMAT*

The RMAT model can only generate graphs with 2^s nodes, where s is an integer scaling parameter. In order to target a fixed number of nodes n_r, we calculate s so that $2^s > n_r$ and delete $2^s - n_r$ random nodes. The choice of the parameters a, b, c, d requires some discussion.

Leskovec et al. [LF07] propose a method to "given a large, real graph [...], generate a synthetic graph that matches its properties". They opt for stochastic Kronecker graphs as the generative method. Starting 2-by-2 stochastic initiator matrix I, Kronecker products are calculated so that I^s is a stochastic matrix of dimension 2^s that yields edge probabilities of a graph. This is equivalent to the RMAT model as it yields the same edge probabilities. They attempt to fit model parameters so that the likelihood that a given original graph O was generated starting from an initiator matrix I is maximized, and propose the kronfit gradient descent algorithm that iteratively estimates I in $O(m)$ time. They do not explicitly mention the case of creating a scaled replica, but it is clear that the method is capable of producing graphs for arbitrary exponents s. We use an implementation of kronfit which is distributed with the SNAP network analysis tool suite. The kronfit algorithm has been employed for other applications, e.g. deriving a distance metric to compare two networks from the estimated initiator matrices [SRK16]. However, we focus on the claim of being able to replicate large networks in a realistic way.

The RMAT generator's fitting method calls the external kronfit tool of the SNAP package [LS14], which tries to fit the initiator matrix probabilities to a given input graph. NetworKit's `RmatGenerator` transparently calls SNAP's `kronfit` if available and receives the estimated parameters. As measurements presented in Fig. 70 show, the kronfit method as implemented in SNAP is rather time-consuming relative to all other fitting methods considered, and applying it to the multi-million edge graphs we want to target seems impractical.

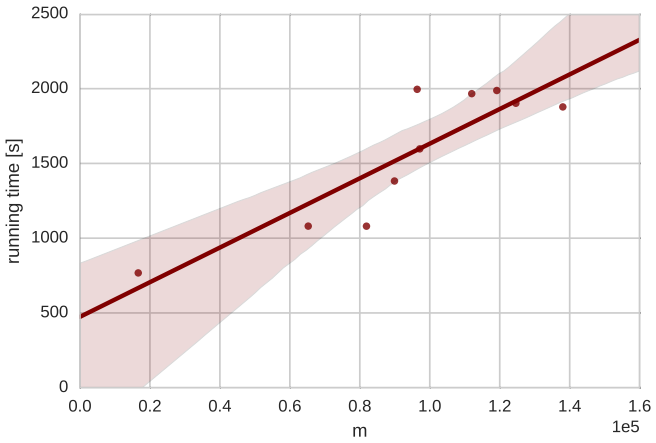


Figure 70: Running time of 50 iterations of the kronfit algorithm in relation to the number of edges m of the input network

The high computational cost of kronfit motivates a closer look at the claim of increased realism as opposed to random guessing of the initiator matrix. Network profiles generated by NetworKit (cf. Sec. 5.3.2) enable a quick exploratory analysis of an original network and replicas created by RMAT, with random parameters on the one hand and parameters generated after 50 iterations of kronfit (a value that the authors propose in [LF07]). The original is `fb-Caltech36`, a small instance from the "Facebook 100" collection of social networks. Fig. 71 shows an excerpt from the generated profiles. The most obvious difference between the original and its two replicas is in the distribution of the local clustering coefficient: While the median is around 0.4 for the original, it is about 0.1 for the replica with random parameters and only slightly higher for the kronfit-generated replica. The same behavior of the generator was observed consistently on other networks. This points to inherent restrictions of generating graphs by stochastic Kronecker multiplication/RMAT that an elaborate parameter fitting does not overcome.

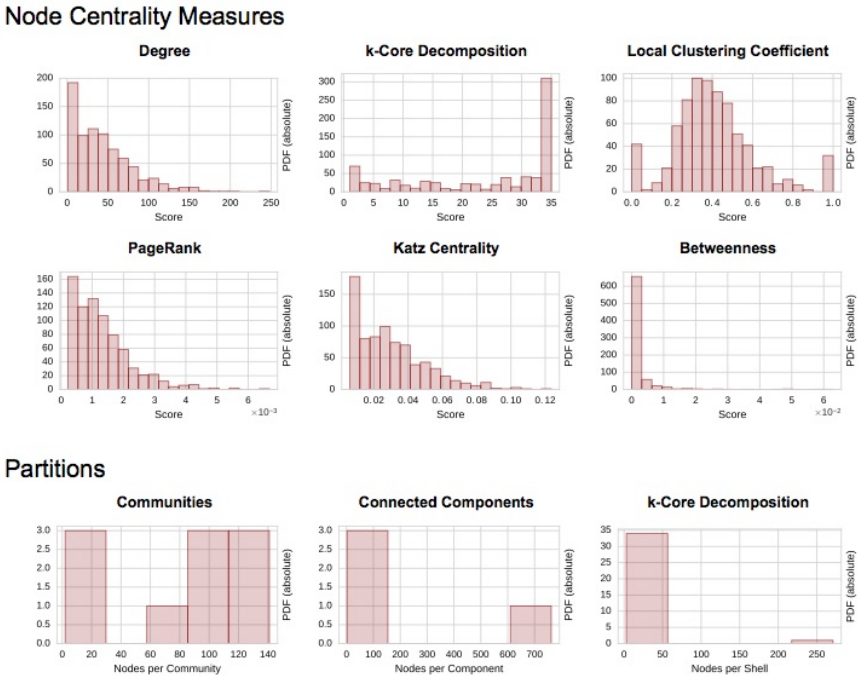
Since running kronfit on every network to be replicated is not practical, we estimate RMAT parameters as follows: We assume that the 100 Facebook networks constitute a class with essential structural commonalities, and run the recommended 50 iterations of kronfit on one typical network, `fb-Caltech36`. The resulting initiator

$$I_{\texttt{fb-Caltech36}} = \begin{pmatrix} 0.378802757 & 0.249474498 \\ 0.255098510 & 0.116624233 \end{pmatrix}$$

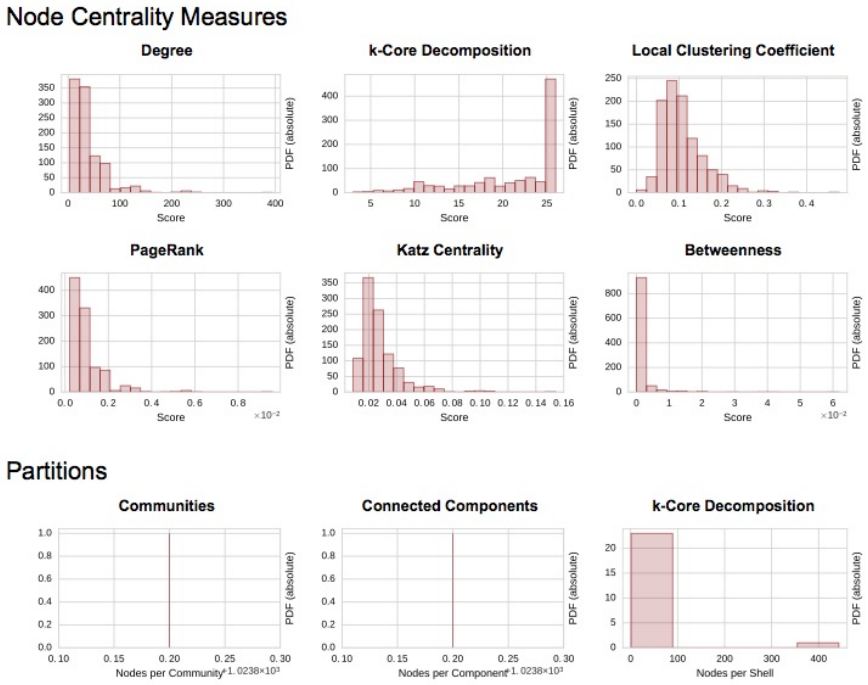
is applied to replicate all Facebook networks. For other sets of networks, the assumption of structural similarity cannot be made, so we use a new random initiator for each replication.

13.5.4 *Hyperbolic Unit Disk*

After consultation with the authors of [vLMP15], we apply the following fitting scheme to the Hyperbolic Unit-Disk Graph generator: The generator receives three parameters, the target number of nodes n', the average degree $\bar{d}$ computed as $2 \cdot (m/n)$, and the power law exponent γ of the degree distribution. The degree power law exponent γ is

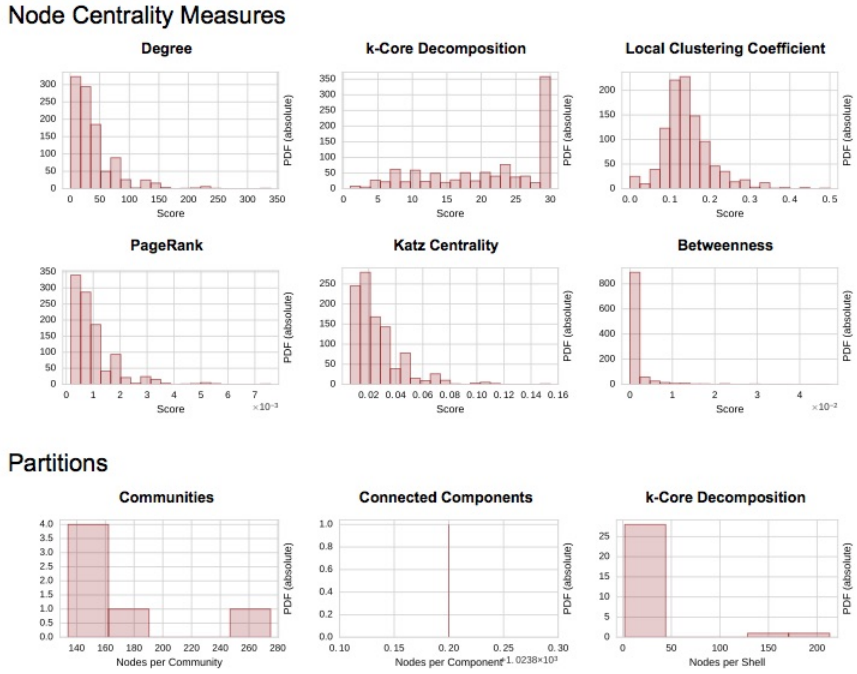


(a) original: `fb-Caltech36`



(b) RMAT replica with random initiator matrix

Figure 71: Structure profiles for the evaluation of the kronfit scheme



(c) RMAT replica with kronfit-generated initiator matrix

Figure 71: (cont.) Structure profiles for the evaluation of the kronfit scheme

estimated as described in Sec. 13.5.1. Note that one restriction of the HUDG generator is that it works only with power law exponents larger than 2. If the power law fit estimates a lower exponent for a given original network, we set it to 2.1. One should note, though, that the HUDG does not necessarily generate a power law distribution that starts at the minimum degree. However, using the high exponents provided by the power law module is most probably still not the best fit. Therefore, further research is needed in order to find a better way to fit power law exponents for replicating networks which is beyond the scope of this work. Nevertheless the minimum exponent of 2.1 is already larger than the exponents we calculate for most networks, so the results are most likely not fundamentally different and also will not change the fundamental characteristics of the networks that are generated by the HUDG so we believe our results are still representative for the HUDG.

13.5.5 *BTER*

$n_d = |\{v \in V : \deg(v) = d\}|$ denotes the number of nodes in G with degree d and $c_d = \frac{1}{n_d} \sum_{u \in V_d} c_u$ denotes the average clustering coefficient for nodes of degree d. According to its original description, BTER receives sequences $(n_d)_{d \in (0,\dots,d_{\max})}$ (the degree distribution) and $(c_d)_{d \in (0,\dots,d_{\max})}$ (the distribution of clustering coefficients per degree) as parameters d and c and recreates them. To replicate a given network O, we simply compute these sequences for O and pass them to BTER. In order to parametrize BTER for scaled replicas, we set d to $(n_d \cdot x)_{d \in (0,\dots,d_{\max})}$, i.e. we multiply the number of nodes of each degree by the scaling factor. This leads to the target number of nodes $n_r = \sum_{d \in (0,\dots,d_{\max})} n_d \cdot x = x \cdot n_o$ while also preserving the general shape of the degree

distribution. We pass c unmodified, which keeps the distribution of clustering coefficients constant with the scaling factor.

13.5.6 *LFR and LFR+*

Fitting LFR and LFR+ requires community detection. PLM refers to the modularity-based community detection heuristic described in Chapter 8.

Given an original network $O = (V, E)$, we produce a scaled replica R^x with $n' = x \cdot n$ nodes as follows:

1. compute the degree sequence $(\deg(u))_{u \in V}$ of O
2. detect communities in O with any suitable method, returning the partition ζ; we use the PLM algorithm
3. compute the sequence $(\lambda_\zeta(u))_{u \in V}$ of local partition coverage scores with respect to ζ
4. concatenation of x copies of ζ; for each subset in ζ, we create $x - 1$ new subsets with the same size

For LFR+ we directly use this degree sequence, the community structure and the local partition coverage scores.

As the original LFR benchmark generates power law distributions for the degree sequence and community size sequence, we fit the parameters of these power law distributions using the distributions of the original network. For the degree, we fit the power law exponent as described above. In theory we could provide the average and maximum degree, too. In practice due to rounding errors the binary search of the minimum degree that is actually performed when the average degree is provided does not work anymore as the actual expected average degree even with a minimum degree of 1 can be higher than the observed one. Therefore we directly provide a power law degree sequence that is generated from the minimum and maximum degree and the calculated exponent. For the community sizes, we also fit the power law exponent in the same way as for the degree distribution and provide the minimum and maximum community size. As many networks in our set of real-world networks contain a few small connected components, the smallest communities usually only contain just 2 nodes while all other communities are much larger. Therefore frequently our minimum power law exponent 1 is chosen as exponent and the average community size is still too low. In these cases we also fit the minimum community size using binary search until the expected average community size is the real average community size.

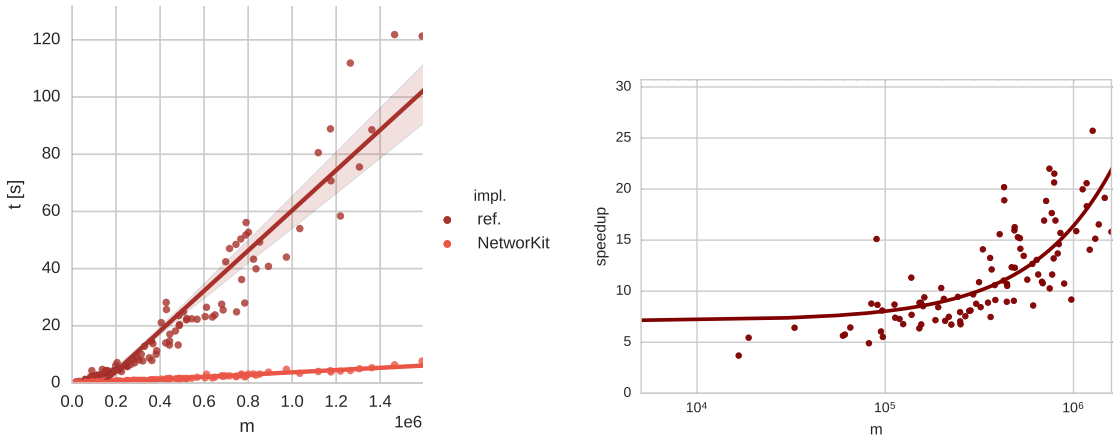
13.6 IMPLEMENTATIONS

While previous sections focused on the generative models, this section discusses aspects of concrete implementations.

13.6.1 *LFR*

A reference implementation of the LFR generator by Fortunato et al. is available online [For]. We created a custom implementation of both the original LFR and the extended

LFR+ model in NetworKit, which is also the first parallelized implementation. Fig. 72 illustrates the speedup that we achieve with a new implementation of the LFR algorithm based on NetworKit. Speed measurements were obtained on the `phipute` machine previously described in this thesis, having 16 physical cores. It yields speedup factors of 5 to 25 for the test set of 100 Facebook social networks, and the factor grows superlinearly with the size of the network to be generated. Our implementation therefore significantly reduces the time needed to generate large synthetic graphs according to the LFR model. The running time difference can be partially traced back to implementation differences: The LFR reference implementation relies on `std::set` to store and test for graph adjacencies, while our implementation uses a simple but apparently more efficient sequential scan on a simple array. Further, as mentioned already, the community graphs are generated in parallell which gives an additional speedup.



(a) absolute time

(b) speedup

Figure 72: Running time measurements of NetworKit implementation of LFR versus reference implementation [For] when replicating a set of networks with m edges

13.6.2 *Other Models*

For the tested generative models, NetworKit includes efficient implementations, such as an implementation of the linear-time algorithms for the Erdös-Renyi and Barabasi-Albert models by Batagelj and Brandes [BB05]. An exception is the BTER model, for which we use the FEASTPACK implementation by Kolda et al. [TGK]. NetworKit implements an adapter class that performs the model fitting and transparently calls the MATLAB-based FEASTPACK binary. Furthermore, as already mentioned, we use the external kronfit tool of the SNAP package [LS14]. NetworKit makes generating replicas of input networks convenient. Each generator class implements a `fit` class method that receives an original graph and a scaling factor and returns a parametrized instance of the generator. Subsequently calling the `generate` method builds a graph.

13.7 EVALUATING THE REALISM OF REPLICAS

In the following we evaluate the realism of (scaled) replicas in different ways.

13.7.1 *Example Replication*

We begin the evaluation of realism by replicating a small example network with each of the models. The input graph is a social network of Bottlenose dolphins [LSB+03], which already served as an example data set throughout this thesis. If a graph is small enough, we can take advantage of graph drawing to inspect network structures. While this is less exact than property measurements, it is also less reductionist. Here we use the ForceAtlas2 layout algorithm implemented in Gephi [Jac09], as well as an additional force attracting nodes to the center of the layout, compacting the layout and preventing components from drifting away. Since the same layout scheme is used for all graphs, this exposes structural differences. Fig. 73 shows a layout of the original network. Node sizes are proportional to degree within one drawing. Consider its basic structure: The graph is connected, with two distinct communities separated by an edge bottleneck (i.e. the graph admits a bipartition cutting only a small bundle of edges).

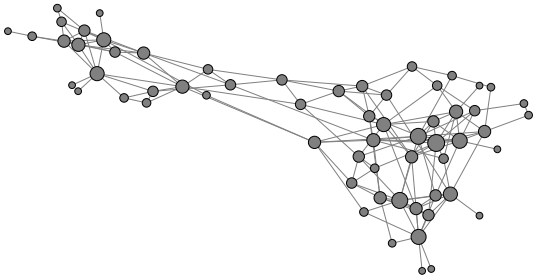
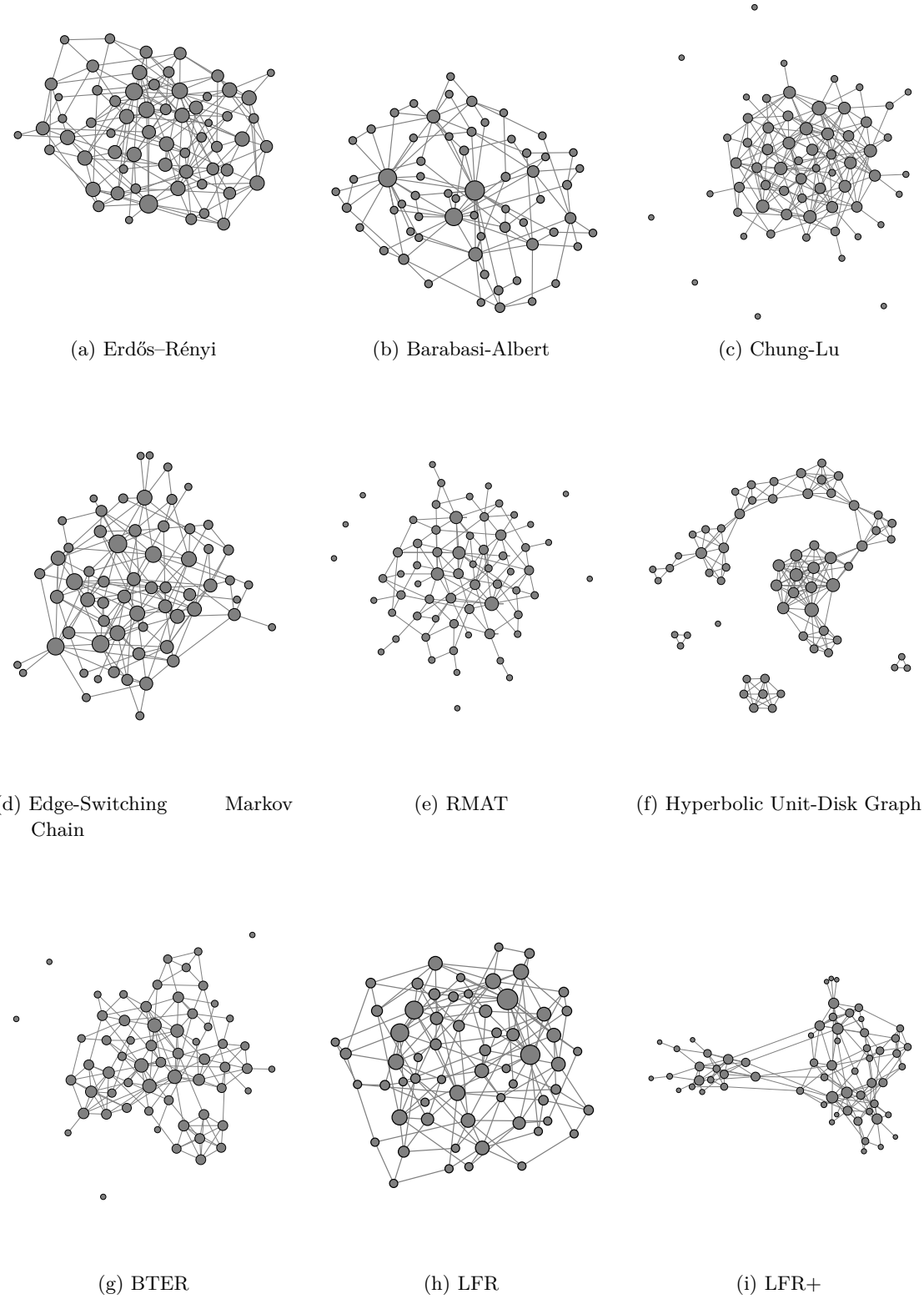


Figure 73: The original dolphins social network

Figure 74 shows layouts for scale-1 replicas of the dolphins network, one for every considered model. Figure 75 repeats this with a scaling factor of 2. We summarize observations as follows: The LFR+ replicas are the only ones commonly recognized as resembling the original. BTER is able to generate a similar community structure, though not matching the original as closely. All other replicas lose the distinctive community structure of the original. The HUDG model creates a kind of community structure, but the graphs have an artificially tubular shape and extreme clustering.

13.7.2 *Replicating Structural Properties*

In the existing literature on generative models, claims of realism are typically substantiated by showing that a set of structural properties is similar for real and synthetic networks. The large palette of properties to choose from and the question which of those properties are essential features makes this a complex problem. An often used approach is to describe a network by a feature vector, a set of scalar properties. These are often maxima, minima or averages of node properties (cf. the structural profiles published by the KONECT project [Kun13b]). This can be reductionist, since these summary values may not give enough information about how the node properties are distributed. We therefore demonstrate how well the different models replicate networks with two types of plots. The first plot type shows scalar properties of the replicas, relative to those of the

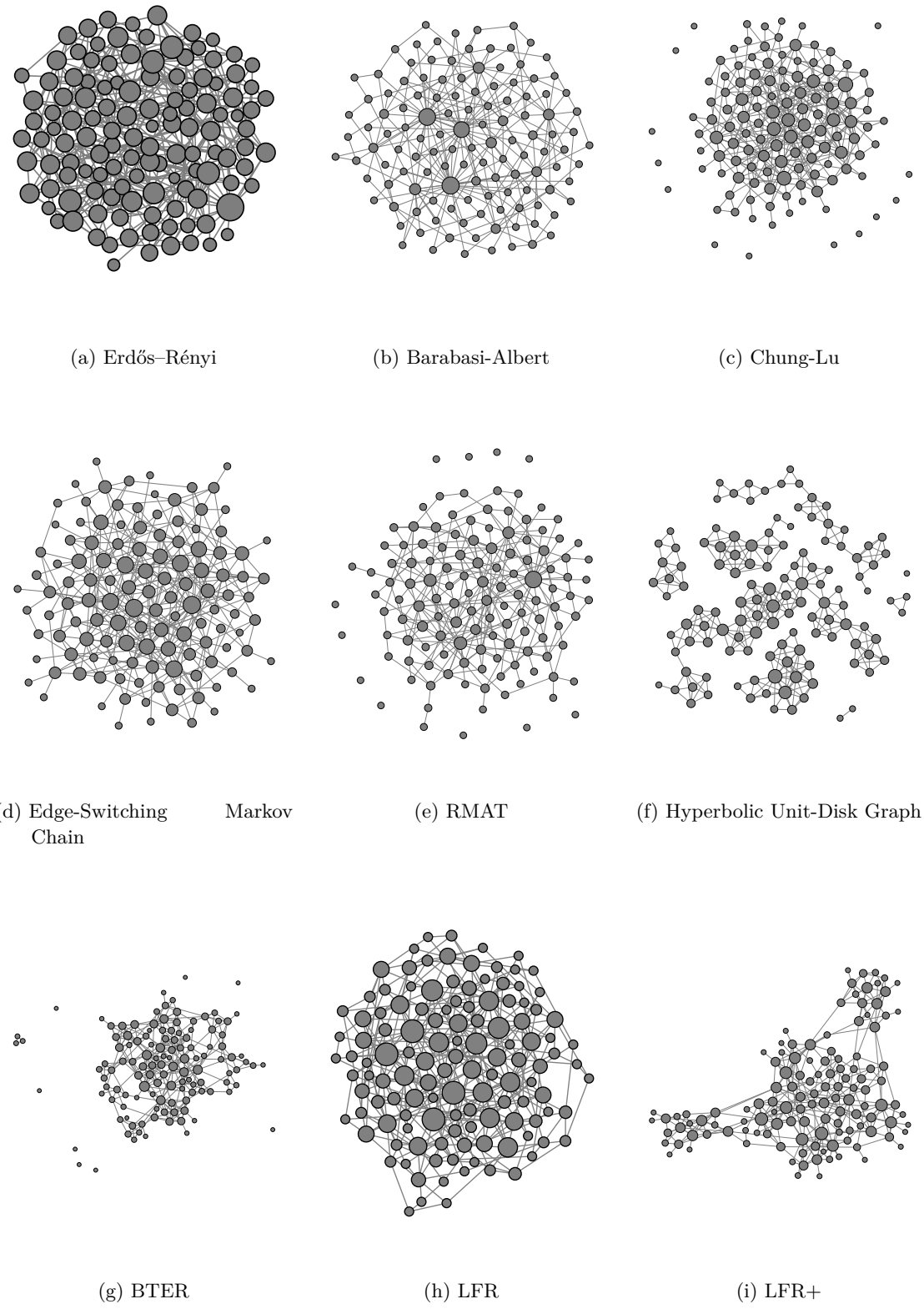


(a) Erdős–Rényi (b) Barabasi-Albert (c) Chung-Lu

(d) Edge-Switching Markov Chain (e) RMAT (f) Hyperbolic Unit-Disk Graph

(g) BTER (h) LFR (i) LFR+

Figure 74: Replicas of the dolphins social network according to different generative models.



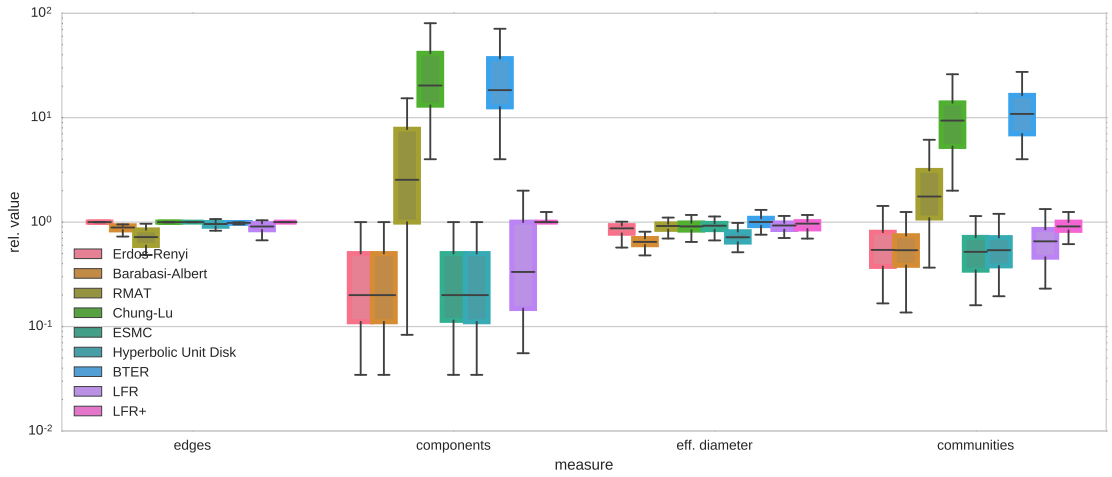
(a) Erdős–Rényi (b) Barabasi-Albert (c) Chung-Lu

(d) Edge-Switching Markov Chain (e) RMAT (f) Hyperbolic Unit-Disk Graph

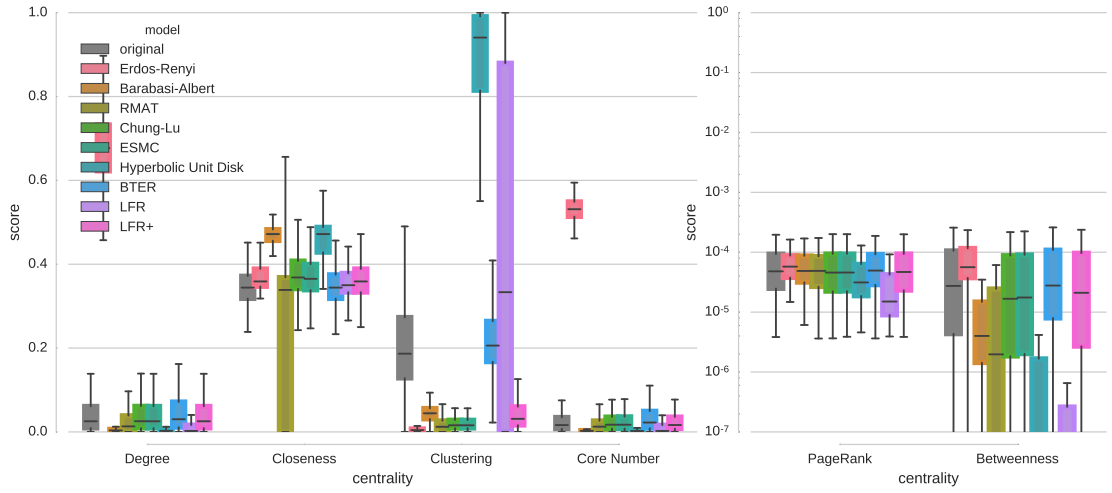
(g) BTER (h) LFR (i) LFR+

Figure 75: Scale-2 replicas of the dolphins social network according to different generative models.

original. Let x_o and x_r denote scalar properties of the original and the replica, respectively. The relative value x_r/x_o is computed for each replica. Box plots depict the distribution of these relative values over the entire set of replicas. Scalar properties included are: the number of edges, the number of connected components, the effective diameter (for 90% of node pairs) of the largest connected component, and the number of communities. The effective diameter is approximated using the ANF algorithm [PGF02]. Communities are detected using the modularity-maximizing algorithm PLM (cf. Chapter 8). This set was chosen so that it could be quickly computed for a large set of networks. The second type of plot covers centrality measures, and is designed to show how the shape of the distributions of node centrality scores of the originals compares to those of the replicas. Each segment of the plot depicts the centrality values of all nodes of all networks in the considered data set or the replicas of a certain algorithm. Since centrality measures from this selection can have very different scales, all centrality scores are normalized to the interval $[0, 1]$.

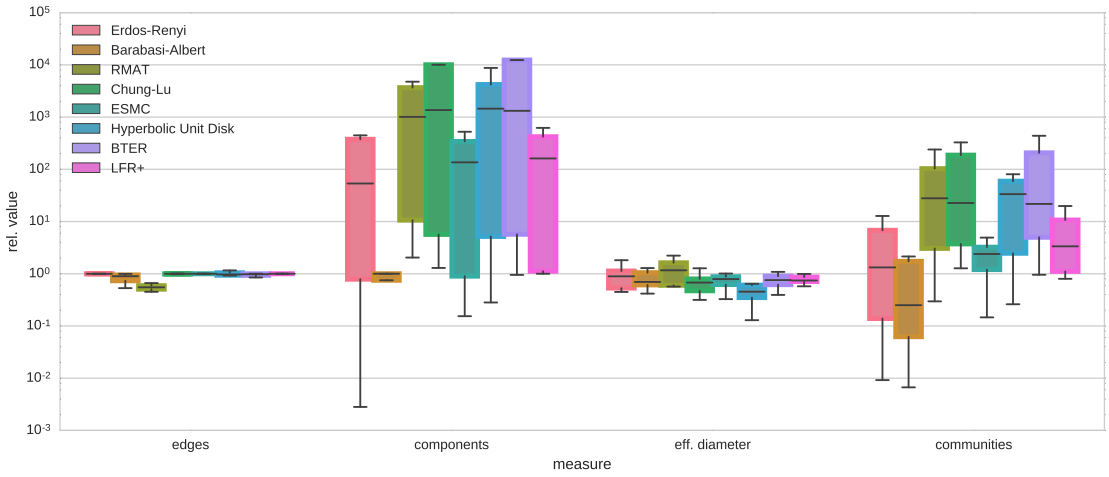


(a) – relative difference of scalar network properties

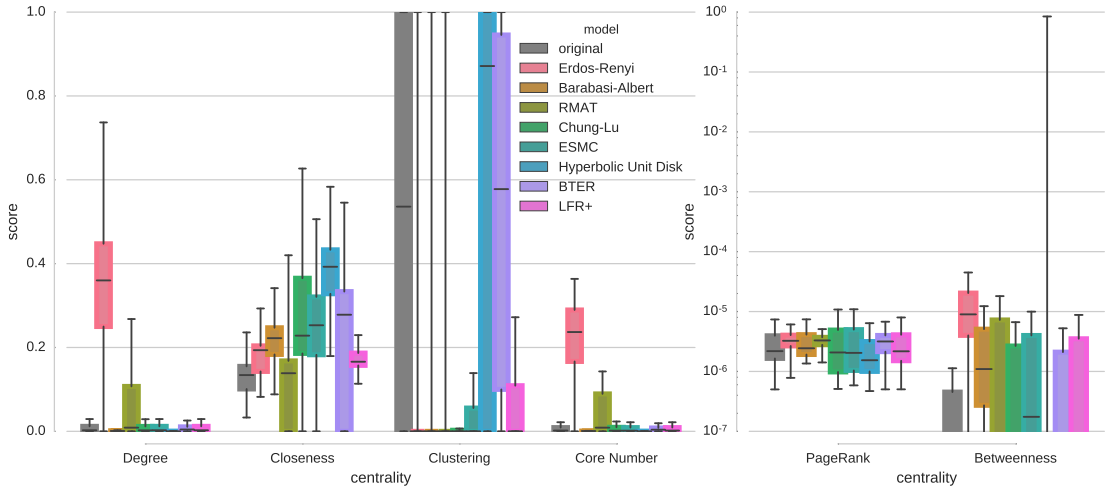


(b) – distribution of centrality scores

Figure 76: Structure replication of Facebook networks



(a) – relative difference of scalar network properties



(b) – distribution of centrality scores

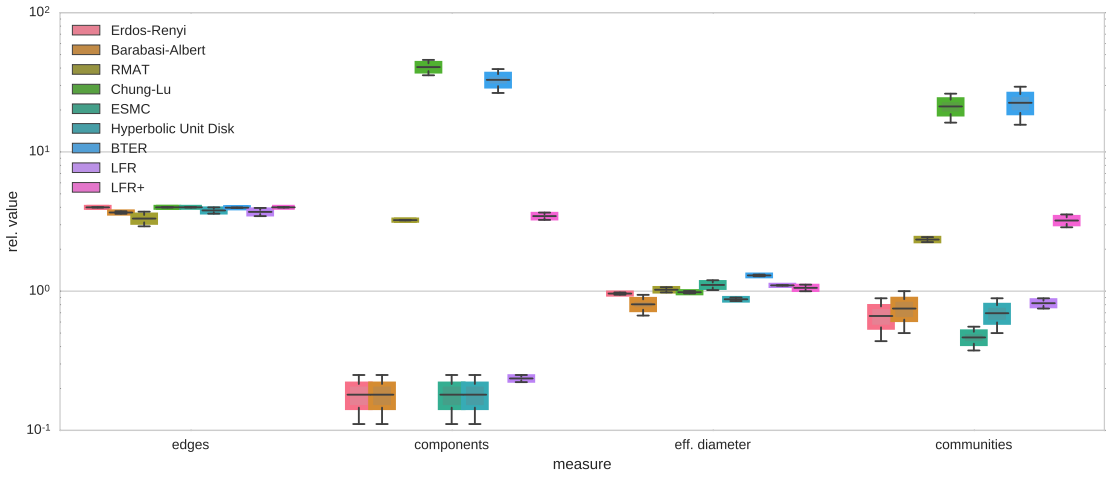
Figure 77: Structure replication of diverse set of networks (second half of Tab. 12)

Describing the results on 100 Facebook networks (Fig. 76a) from left to right, we observe the following results for the measured scalar properties: All models were parametrized to produce exactly $n' = n$ nodes, while some degree of freedom exists in the number of edges. However, all generators replicate the number of edges with a narrow variance, which is largest for RMAT. LFR+ is the only model that matches the number of components with high accuracy, while it is lower for ER, BA, ESMC, HUDG and vanilla LFR, and significantly higher for CL and BTER. The latter can probably be explained by the generation of isolated nodes instead of degree-1 nodes which we have already seen in the visualizations before. There is no extreme deviation in terms of the effective diameter, but note that for small world networks, relatively small differences in the effective diameter may indicate significant structural differences. The Barabasi-Albert and Hyperbolic Unit Disk models deviate the most and produce lower diameters. LFR+, which receives a partition of the original into communities as input, replicates it closely, but all yield different numbers of communities. The Chung-Lu and BTER generators increase the number of communities by a factor of 10 on average which can be explained by the larger number of connected components. Overall, LFR+ emerges with the most accurate replicas from this experiment.

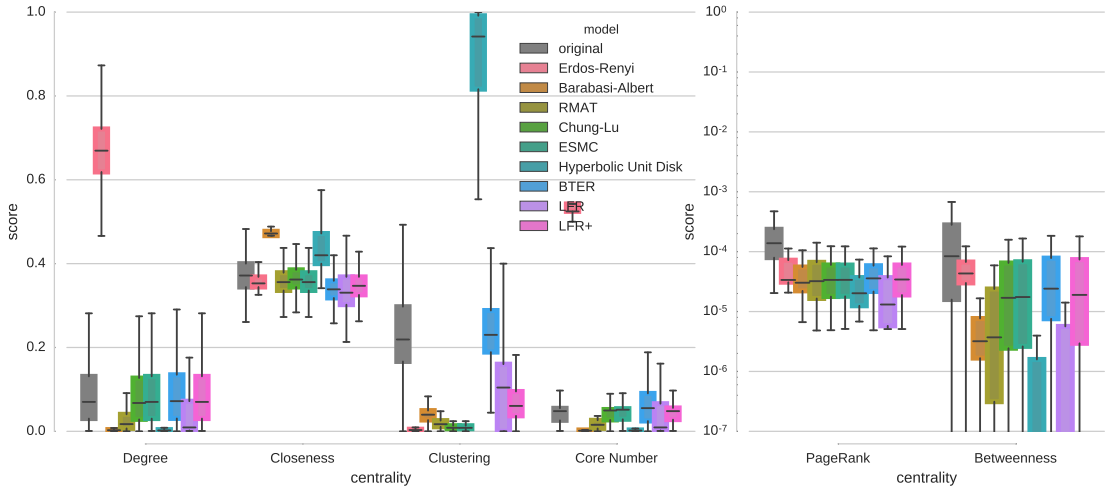
The distributions of node centralities (Fig. 76b) compare as follows: All models except ER are capable of producing skewed degree distributions, with CL, ESMC, BTER and LFR+ matching the original closely. Closeness is approximately matched by most models, but BA, RMAT and HUDG deviate significantly. The original networks feature a wide range of clustering coefficients, and only the BTER model, which receives explicitly this distribution, matches them exactly. For HUDG, clustering is extremely artificially high with a median close to 0.9, while LFR produces an unrealistically large variance. For the k-core numbers, random graphs are clearly outliers. RMAT, CL, ESMC, BTER and LFR+ match well, while the very narrow distributions of the others point to a lack of differentiated k-core structure. Only small variations exist with respect to PageRank, and HUDG and vanilla LFR have strong deviations in terms of betweenness centrality, but interpretation is not straightforward in this case. In summary, extreme deviations in centrality score distributions clearly give away the artificiality of some synthetic graphs, such as the clustering coefficient for HUDG or the degree distribution for ER. Other differences are more subtle, but possibly relevant. BTER replicates centralities most accurately.

The results on a different set of networks (second half of Tab. 12) are presented in Fig. 77. A notable difference is the extreme variance of clustering coefficients in the original set, which LFR+ cannot replicate. Again LFR+ performs well for the majority of properties.

Fig. 78 shows results of a repetition of the experiment with a scaling factor of 4. All models except RMAT achieve the targeted edge factor of $m' = 4 \cdot m$. The number of components is unrealistically increased by RMAT and CL. For the effective diameter, small relative differences matter: BA and HUDG model tend to create smaller worlds than reality. LFR+ produces a remarkably exact match, considering that the generator does not explicitly target the diameter. It does however target a higher number of communities, which is desired and achieved. LFR keeps the number of communities constant on average, while many other models produce fewer communities than the originals. The relative differences in the distributions of centralities are qualitatively equivalent to those in Fig. 76b.



(a) – relative difference of scalar network properties

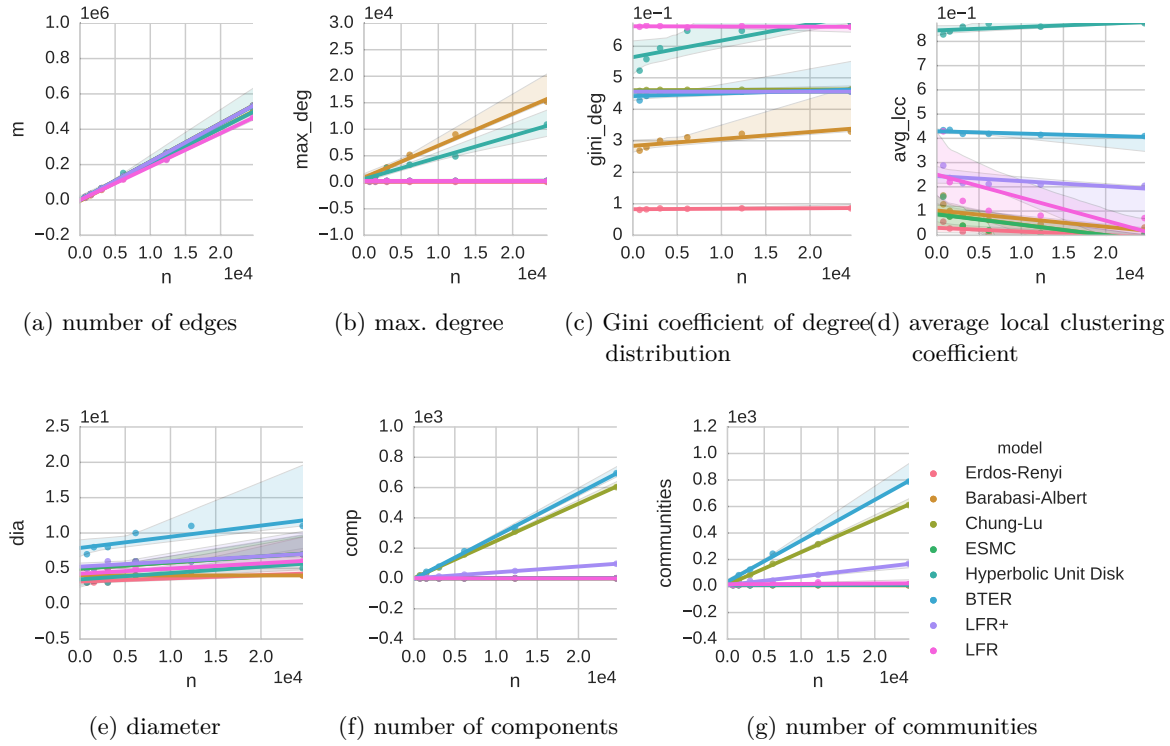


(b) – distribution of centrality scores

Figure 78: Structure replication of Facebook networks with scaling factor 4

13.7.3 *Scaling Behavior of Generators*

The following experiments consider the scaling behavior of generative models. Given the parametrization discussed before, we look at the evolution of structural features with growing scale factor x up to $x = 32$. We consider basic scalar features, including the number of edges, number of connected components, diameter of the largest connected component, shape of the degree distribution as measured by the maximum degree and the Gini coefficient [Gin12], average local clustering coefficient and number of communities.

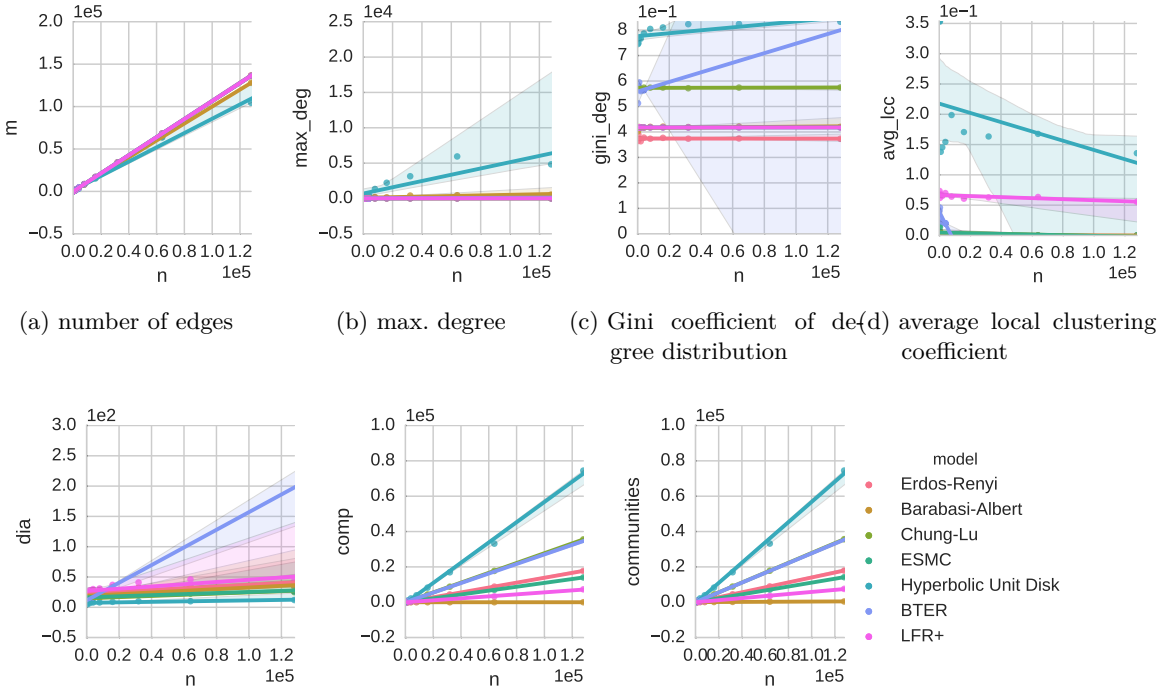


(a) number of edges (b) max. degree (c) Gini coefficient of degree distribution (d) average local clustering coefficient

(e) diameter (f) number of components (g) number of communities

Figure 79: Scaling behavior of the different generators on the `Caltech36` network.

Figure 79 shows the results of the scaling experiments for the `Caltech36` network. The number of edges of the replicas of the `Caltech36` network is increased almost linearly by all generators to about 500 thousand edges which approximately corresponds to 32 times the edges of the original network. Therefore all generators seem to keep the average degree of the original network which is actually not completely realistic according to our scaling study on the 100 Facebook networks, here we observed a slight increase of the average degree. As the average degree or all degrees are input parameters of all of the generators and we did not scale them. This is expected, though. It is surprising therefore that the maximum degree strongly increases up to 10 or 15 thousand with the hyperbolic unit disk generator and the Barbarasi-Albert generator respectively. The original maximum degree is 248, so this is even more than 32 times the maximum degree of the original network (which is about 8000). From scaling the study on the 100 Facebook networks we could expect an increase, but rather in a lower range, so the degree distribution of the Barbarasi-Albert and the hyperbolic unit disk generator are not realistic. Concerning the Gini coefficient one can clearly see that ER does not generate a skewed degree distribution



(a) number of edges (b) max. degree (c) Gini coefficient of degree distribution (d) average local clustering coefficient

(e) diameter (f) number of components (g) number of communities

Figure 80: Scaling behavior of the different generators on the Colorado Springs epidemiological contact network.

at all. All generators that get the exact degree sequence as input keep the gini coefficient constant, which is also relatively realistic from our scaling study.

The average local clustering coefficient of 0.43 in the original network is almost exactly reproduced by BTER, which is not suprising as this is an input parameter. The HUDG generator generates a very high clustering coefficient of 0.8 as already seen in the structure replication experiments. Our new LFR+ generator is not that far off with 0.25 and a slightly decreasing clustering coefficient, which is actually realistic as we saw in the scaling behavior of the 100 Facebook networks. While the LFR generator can reproduce the clustering coefficient initially, the clustering coefficient decreases down to less than 0.1 at a scaling factor of 32. The other generators fail to replicate the clustering coefficient at all scales, they generate much lower clustering coefficients.

The original diameter of 6 is almost exactly kept by our new generator LFR+, all other generators except BTER generate networks with slightly lower diameters while BTER generates networks with almost twice the diameter. All generators show a slight increase of the diameter when the networks are larger which is consistent with our scaling study on the 100 Facebook networks.

Concerning the number of components and the number of communities one can see again that Chung-Lu und BTER generate a large number of components which is probably due to the large number of degree-0 nodes. The original network consists of a giant component and 3 small components, it is therefore not surprising that most generators do not generate multiple connected components while LFR+ scales them linearly, which is due to its parametrization. The number of communities grows similarly. The original network is split in 11 communities, 8 of them are therefore non-trivial. We would therefore expect 352 communities in the scaled version, but PLM finds only about 200 but actually with all other generators much less communities can be found so LFR+ seems to generate the most realistic community structure. BTER also seems to generate non-trivial communities that can be found, while there are around 7000 connected components there are around 8000 communities founds which indicates at least 1000 non-trivial communities – which is not realistic, either.

The results on the Colorado Springs network (see 13.7.4 for explanation) in Figure 80 are similar but there are a few differences. The vanilla LFR had to be excluded because it had extreme running times for this relatively small network, likely because the original LFR algorithm has problems satisfying the estimated parameters for this network with untypical community structure. The maximum degree of Barbarasi Albert is much smaller and its Gini coefficient is also similar to other generators.

All in all the LFR+ generator is the only generator that keeps the degree distribution, and produces a realistic clustering coefficient and a small diameter while keeping the graph connected and preserving a moderate number of communities. All other generators are either unable to keep the diameter or the connectivity or the number of communities.

13.7.3.1 *Replicating Running Times of Graph Algorithms*

Synthetic graphs are frequently used in algorithm engineering to estimate the running time of an algorithm experimentally. A typical (but often implicit) argument is that the synthetic graphs used are sufficiently similar to real inputs so that the reported running times are representative for the expected running times in real-world applications. In the following, we examine this argument experimentally: Given a set of real complex networks, we use the generative models to produce a corresponding set of synthetic graphs,

each a "replica" of a real one obtained by fitting the model (described in Sec. 13.5). Then a variety of graph algorithms is run on both the original set and the replica sets. The set of algorithms is selected to cover a variety of patterns of computation and data access, each of which may interact differently with the graph structure. It consists of algorithms for connected components (essentially breadth-first search), PageRank (via power iteration), betweenness approximation (according to Geisberger et al. [GSS08], essentially a number of breadth-first search passes), community detection (PLM, cf. Ch. 8), core decomposition (according to [DRZ14]), triangle counting (according to [HLM+16]), and spanning forest (essentially Kruskal's algorithm without edge weights). Some of the algorithms are executed in parallel, but running time replication results were qualitatively very similar when running them sequentially.

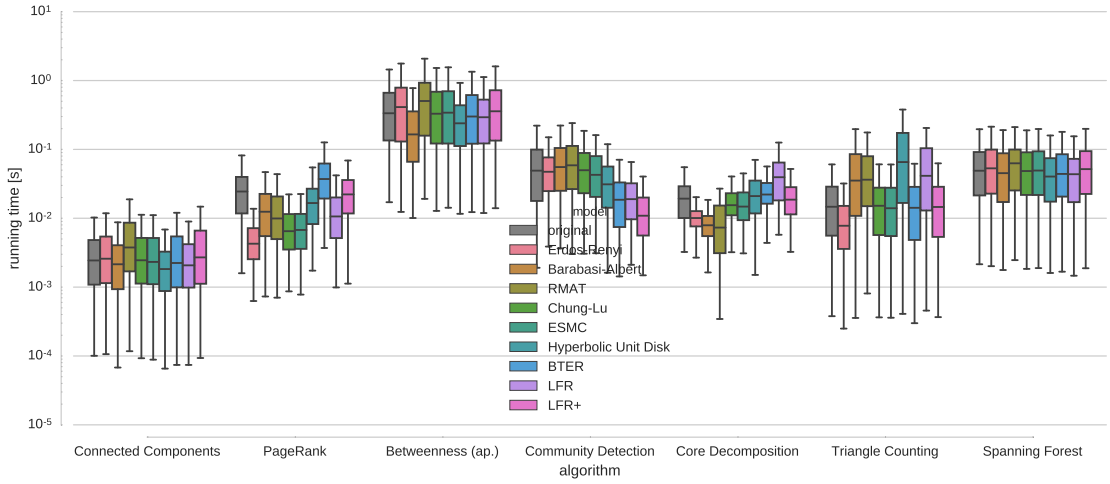
If the argument above holds, then the running times on the replica set should, statistically speaking, match the running times on the original set. We see that they diverge in nontrivial ways (see Fig. 81): The gray segments of the box plots represent the distribution of running times measured on a set of original networks. Ideally, the distribution on the synthetic networks would be identical. First we take measurements on the set of Facebook networks (Fig. 81a) and then repeat them on a set of larger networks (the first half of Tab. 12) for higher running times with less random fluctuation (Fig. 81b) (here, the slowest implementation, BTER, had to be excluded due to high running times of the generator).

network	**type**	n	m
con-fiber_big	connectome	591428	46374120
as-22july06	internet topplogy	22963	48436
coAuthorsDBLP	scientific coauthorship	299067	977676
actor-collaboration	movie actor collaboration	382219	15038083
Lastfm	online social network	1193699	4519020
Foursquare	online social network	639014	3214986
email-Enron	email communication	36691	183830
PGPgiantcompo	PGP web of trust	10680	24316
as-22july06	internet topology	22963	48436
hep-th	scientific coauthorship	8361	15751
dolphins	animal social network	62	159
power	power grid	4941	6594
cnr-2000	web graph	325557	2738969

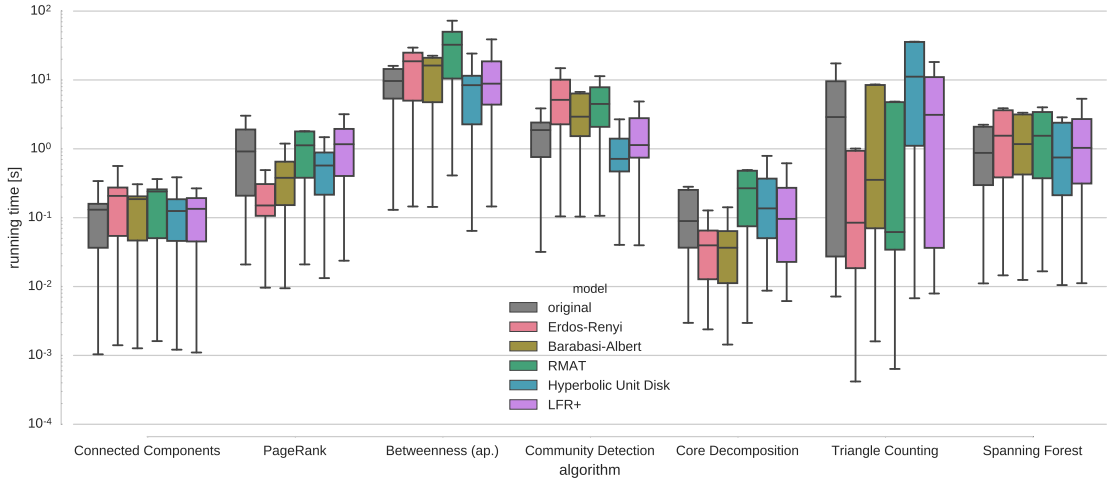
Table 12: Additional networks used

Little variance between the models exists for connected components and spanning forest computation, since their running time is nearly solely based on the number of edges. Other algorithms show how much running time can depend on network structure, especially community detection, core decomposition, triangle counting and PageRank. Convergence for PageRank has been shown to depend on spectral properties of the graph. In general, the running time measurements obtained on LFR+ match the originals closely in most cases. Surprisingly, an exception is community detection on the Facebook net-

works, since LFR+ explicitly replicates community sizes, but the distribution matches again closely for the larger networks. BTER shows close matches as well.



(a) – Facebook networks



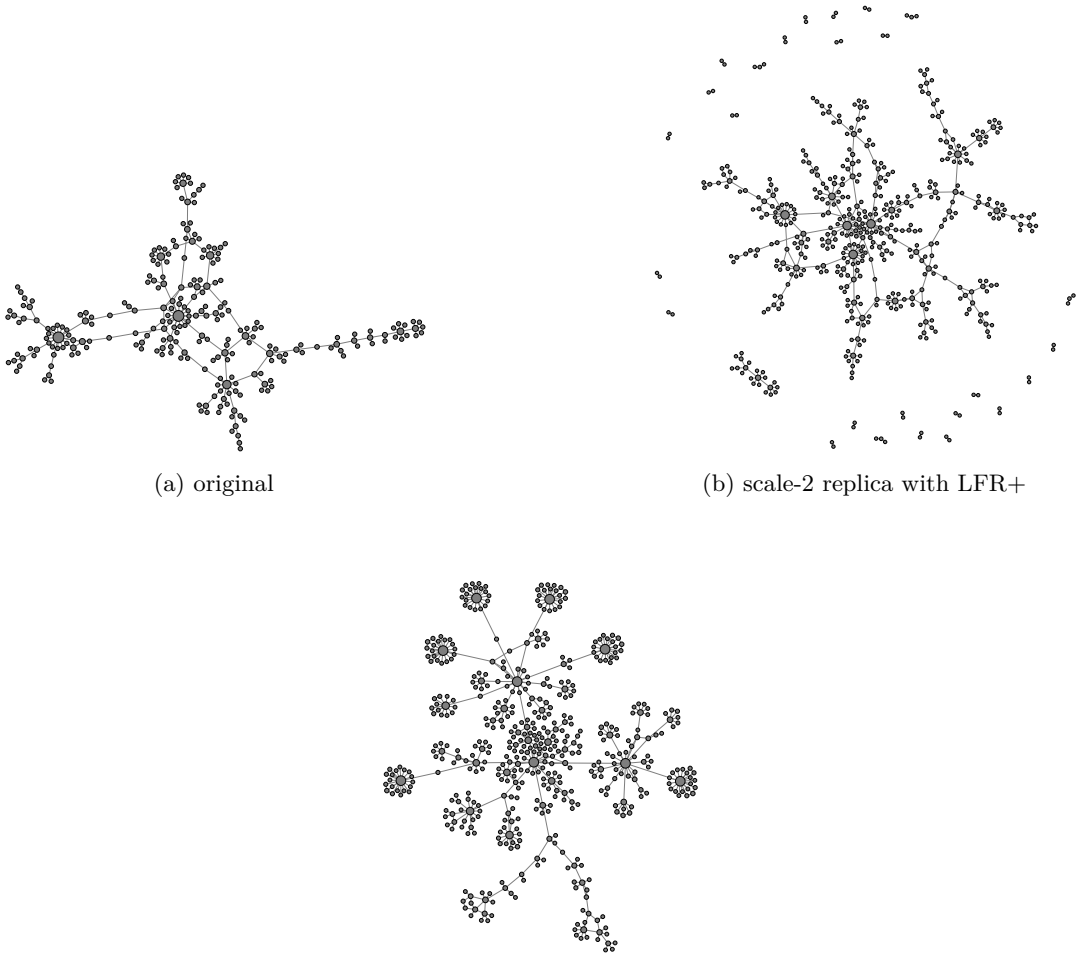
(b) – mixed set of larger complex networks

Figure 81: Running time replication

13.7.4 *An Additional Case Study*

An epidemiological network frequently used in studies on the transmission of HIV is based on data collected in Colorado Springs by Potterat et al. [PPPM+02]. It contains 250 individuals who were in contact in the 1980s through sex or injection drug use. Figure 82a shows a force-directed layout of the network's graph. Characteristic for this network is its tree-like structure and the presence of high-degree hubs with attached "satellite" nodes of degree 1. The Colorado Springs network is an instance of network data that cannot be shared freely due to legal restrictions, but no such restrictions apply to a synthetic replica. A replica made by the LFR+ generator reproduces structural features of the original with the highest accuracy among the considered models. More importantly,

scaled replicas retain these essential properties, including the hub-and-satellite structure. Figure 82b shows the network replicated with a scaling factor of 2, which is remarkably close to the original. Like several other generators, LFR+ generates additional small connected components as an artifact. If this is undesirable, a postprocessing step could prune the network down to its giant connected component containing a large majority of nodes. (Interestingly, the original data set contained many singletons and isolated dyads that were removed from the graph in a preprocessing step).



(a) original

(b) scale-2 replica with LFR+

(c) 500 node sample from a scale-200 000 replica with LFR+

Figure 82: Colorado Springs epidemiological contact network

Real epidemiological contact network data is difficult to collect, further complicated in the case of HIV by sex and drugs being tabu subjects. This makes obtaining such a network on the scale of an entire population impractical. In such a scenario, the ability to create realistic large synthetic replicas of smaller real networks may be highly relevant. As an explorative case study, we let LFR+ generate a replica of the Colorado Springs network with 50 million nodes, which corresponds to a scaling factor of 200 000. Fig. 82c shows a sample from a $5 \cdot 10^7$ million node replica. This network's structure is quite

different from the clustered "friendship networks" of dolphins and humans (Facebook). Remarkably, LFR+ also replicates many aspects of the original's structure very closely, such as a tree-like structure with hubs and attached satellites. These aspects are retained even for huge scaling factors. This makes LFR+ a promising candidate to deliver large data sets in cases where large amounts of real data are likely unobtainable. Further domain-specific validation of the suitability of such replicas would be interesting.

13.8 PERFORMANCE

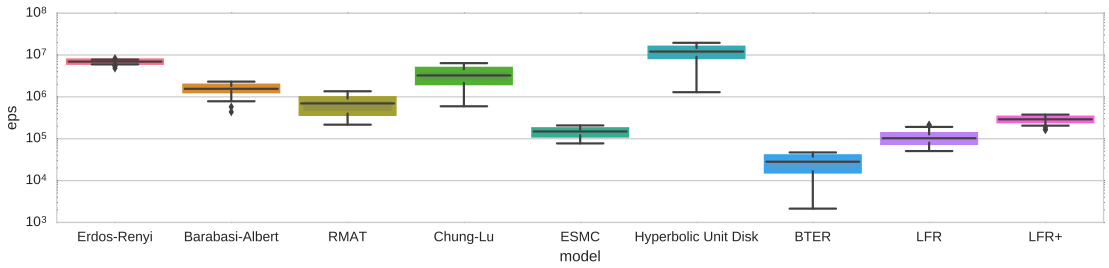


Figure 83: Fitting and generating: processing speed measured in edges/s (size of replica graph / total running time, measured on 100 Facebook graphs)

Fig. 83 shows how long it takes to fit the various generators an input network and generate a replica. Processing speed given in edges per second to express absolute running time normalized by the size of the generated network. The entire set of Facebook networks was used to generate the measurements, so generated replicas ranged from about 15 000 to 1.5 million edges. For all models, generating the graph takes up the vast majority of time. BTER's MATLAB-based implementation is slowest, while the ER and HUDG generators are the fastest (however, we need to keep in mind that the HUDG generators running time grows superlinearly). Our implementation of LFR and LFR+ is not among the fastest generators, but fast enough to produce millions of edges in minutes. It is not highly optimized, so future work could focus on processing speed gains.

13.9 CONCLUSION

We have presented a new generator LFR+ for replicating and scaling existing networks which is based on the popular LFR benchmark but uses a more flexible parametrization in order to capture more properties of the original network. In an extensive experimental evaluation we have shown that it is capable of generating networks which are both similar to the original network in terms of measures like diameter or centrality distribution and leads to similar running times of network analysis algorithms. Using LFR+ it is possible to realistically replicate an existing network, and to scale the synthetic version by orders of magnitude, e.g. in order to test algorithms on larger data sets where they are not available due to restrictions like being proprietary or sensitive. Furthermore it allows to create anonymized copies of such networks that can be distributed as they no longer contain the original data but are still structurally similar and thus allow to conduct experiments on them. While other generators sometimes perform better concerning certain criteria,

none of the other generators is capable of approximately reproducing such a wide range of properties and running times.

The BTER generator is especially good at reproducing clustering coefficients (which are given as input), but it is both slower in its current implementation (provided by the authors) and unable to keep other properties like the characteristically low diameter while scaling the network.

The HUDG generator produced artificially extreme clustering. In defense of the general approach we mention that the extended model of Kriokouv et al. [KPK+10] provides and additional parameter T, yielding an adjustable clustering coefficient in practice. When varying T between 0 and 1, the clustering coefficient obtains values between 0 and 0.8. Implementations for this extended model are available from Aldecoa et al. [AOK15] (with prohibitive quadratic time complexity), and a preprint from Looz et al. [vLM15] describes an $O((n^{3/2} + m) \log n)$ implementation.

The RMAT generator – which was previously presented as a method for replicating networks [LF07] – could not convince in our experiments since it does not closely and comprehensively reproduce the properties of the original network, even if the comparatively expensive kronfit algorithm is applied to fit parameters. The claim that an RMAT replica can even serve as an "anonymized" substitute for the original network needs to be called into question.

While our generator can already convince in many aspects, it is also clear that there is still room for improvements. Especially the clustering coefficients should be addressed in future work. As our generator uses random subgraphs with fixed degree sequence as building blocks, replacing them by graphs with higher clustering coefficients could be a good approach. Using BTER for this step might be a good starting point, but is definitely not that easy as BTER does not exactly reproduce the degree sequence. Therefore a more sophisticated approach like a modified version of BTER or some post-processing step are definitely needed.

CONCLUSION OF PART V

Part V focused on generative models for realistic synthetic graphs. Related work has proposed a variety of models, some of them with claims of comprehensive realism, i.e. matching patterns commonly observed in real complex networks. However, defining and quantifying realism is a complicated task. We explore two different approaches of quantification: On the one hand, a replica should match a catalog of structural properties of the original, e.g. the diameter or distributions of node centrality measures. On the other hand, a replica should be a good substitute for the original when performing running time measurements during algorithm engineering. We therefore evaluate whether the running times for various graph algorithms measured on synthetic graphs are representative for those observed on the original networks.

Beyond the goal of creating replicas of the same size, in this work we specifically target the use case of producing a scaled replica of a given original network, which has so far not received much attention in the literature. We considered the scaling behavior of real networks in an explorative study to define desired scaling properties. We then propose suitable fitting schemes which parametrize the considered models in order to generate a scaled-up version of the input graph.

As the most promising approach for both goals, we propose the LFR+ generator, a modification of the LFR model for community detection benchmarks. We harness the true flexibility of LFR's algorithmic methods to increase the realism of the replication. Our fast implementation in NetworKit generates graphs according to the plain LFR and extended LFR+ model and does so significantly faster than the reference implementation, also by introducing parallelism. LFR+ improves on its predecessor LFR in terms of flexibility, realism, and efficiency of implementation. Specifically, LFR+ is generally more realistic than the RMAT, BTER and Hyperbolic Unit Disk Graph models, all of which have been proposed as realistic to the point of yielding suitable substitutes of real network data. In contrast to the kronfit algorithm for RMAT, the model fitting scheme for LFR+ is fast and applicable to larger networks. We show that our design yields a scalable and effective tool for replicating a given network – and possibly scale it by orders of magnitude – while closely preserving important properties on the micro- and macro level. This yields realistic test data for the engineering of computational methods on networks where suitable real data is not available.

You can stand on the shoulders of giants. Or a big enough pile of dwarfs. Works either way.
– Discordian wisdom

[AB02] R. Albert and A.L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[ABP14] Jeff Alstott, Ed Bullmore, and Dietmar Plenz. powerlaw: a python package for analysis of heavy-tailed distributions. *PLoS ONE*, 9(1):e85777, 2014.

[ACG+09] Frederico AC Azevedo, Ludmila RB Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata EL Ferretti, Renata EP Leite, Roberto Lent, Suzana Herculano-Houzel, et al. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541, 2009.

[ACL00] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 171–180. Acm, 2000.

[ACL06] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *FOCS '06: Proc. of the 47th Annual IEEE Symp. on Foundations of Computer Science*, pages 475–486, Los Alamitos, CA, USA, October 2006. IEEE Computer Society.

[AG] I. Safro A. Gutfraind, L.A. Meyers. Musketeer: Multiscale entropic network generator.

[AL06] R. Andersen and K. Lang. Communities from seed sets. In *Proc. of the 15th Int'l Conf. on World Wide Web*, page 232. ACM, 2006.

[AM11] Rodrigo Aldecoa and Ignacio Marín. Deciphering network community structure by surprise. *PloS one*, 6(9):e24195, 2011.

[ANK14] Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. Network sampling: From static to streaming graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(2):7, 2014.

[AOK15] Rodrigo Aldecoa, Chiara Orsini, and Dmitri Krioukov. Hyperbolic graph generator. *arXiv preprint arXiv:1503.05180*, 2015.

[Apa15a] Apache. Website of the framework Apache Flink: https://flink.apache.org/, April 2015.

[Apa15b] Apache. Website of the framework Apache Giraph: http://giraph.apache.org/, April 2015.

[ATK+11] Tharaka Alahakoon, Rahul Tripathi, Nicolas Kourtellis, Ramanuja Simha, and Adriana Iamnitchi. K-path centrality: A new centrality measure in social networks. In *Proceedings of the 4th Workshop on Social Network Systems*, page 1. ACM, 2011.

[Ave13] Avery Ching. Scaling apache giraph to a trillion edges. https://www.facebook.com/notes/facebook-engineering/scaling-apache-giraph-to-a-trillion-edges/10151617006153920, 2013. [Online; accessed 30-July-2014].

[BA99] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[Bag08] J.P. Bagrow. Evaluating local community methods in networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008:P05001, 2008.

[BB05] Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Physical Review E*, 71(3):036113, 2005.

[BBC+11] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[BCH+15] Michele Borassi, Pierluigi Crescenzi, Michel Habib, Walter A. Kosters, Andrea Marino, and Frank W. Takes. Fast diameter and radius bfs-based computation in (weakly connected) real-world graphs: With an application to the six degrees of separation games. *Theoretical Computer Science*, 586:59 – 80, 2015. Fun with Algorithms.

[BCSV04] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.

[BDG+08] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE Trans. Knowledge and Data Engineering*, 20(2):172–188, 2008.

[BEH+10] Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke. Nephele/pacts: A programming model and execution framework for web-scale analytical processing. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 119–130, New York, NY, USA, 2010. ACM.

[BFM14] Michel Bode, Nikolaos Fountoulakis, and Tobias Müller. The probability that the hyperbolic random graph is connected. 2014. Preprint available at http://www.staff.science.uu.nl/~muell001/Papers/BFM.pdf.

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[BHJ09] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: an open source software for exploring and manipulating networks. In *International Conference on Weblogs and Social Media*, pages 361–362, 2009.

[BK14] Lars Backstrom and Jon Kleinberg. Romantic partnerships and the dispersion of social ties: a network analysis of relationship status on facebook. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*, pages 831–841. ACM, 2014.

[BM04] Vladimir Batagelj and Andrej Mrvar. *Pajek—analysis and visualization of large networks*, volume 2265 of the series Lecture Notes in Computer Science pp 477-478. Springer, 2004.

[BM15] Elisabetta Bergamini and Henning Meyerhenke. Fully-dynamic approximation of betweenness centrality. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 155–166, 2015.

[BMS+14a] David A. Bader, Henning Meyerhenke, Peter Sanders, Christian Schulz, Andrea Kappes, and Dorothea Wagner. Benchmarking for graph clustering and partitioning. In *Encyclopedia of Social Network Analysis and Mining*, pages 73–82. 2014.

[BMS+14b] Aydin Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent advances in graph partitioning. *arXiv preprint arXiv:1311.3144*, 2014. Preprint: http://arxiv.org/abs/1311.3144.

[BMS15] Elisabetta Bergamini, Henning Meyerhenke, and Christian Staudt. Approximating betweenness centrality in large evolving networks. In *Proceedings of the Seventeenth Workshop on Algorithm Engineering and Experiments, ALENEX 2015, San Diego, CA, USA, January 5, 2015*, pages 133–146, 2015.

[BMSW13] David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors. *Graph Partitioning and Graph Clustering*. Number 588 in Contemporary Mathematics. 2013.

[Bra01] U. Brandes. A faster algorithm for betweenness centrality. *J. Mathematical Sociology*, 25(2):163–177, 2001.

[Bra10] L. Karl Branting. Incremental detection of local community structure. In *Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '10, pages 80–87, Washington, DC, USA, 2010. IEEE Computer Society.

[Bra16] Ulrik Brandes. Network positions. *to appear in Methodological Innovations*, 2016.

[BRMW13] Ulrik Brandes, Garry Robins, Ann McCranie, and Stanley Wasserman. What is network science? *Network Science*, 1(01):1–15, 2013.

[BRV11] Paolo Boldi, Marco Rosa, and Sebastiano Vigna. HyperANF: Approximating the neighbourhood function of very large graphs on a budget. In *Proceedings of the 20th international conference on World wide web*, pages 625–634. ACM, 2011.

[BS09] Ed Bullmore and Olaf Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature Reviews Neuroscience*, 10(3):186–198, 2009.

[BS11] C.E. Bichot and P. Siarry. *Graph Partitioning*. ISTE. Wiley, 2011.

[BS13] Sanjukta Bhowmick and Sriram Srinivasan. A template for parallelizing the louvain method for modularity maximization. In *Dynamics On and Of Complex Networks, Volume 2*, pages 111–124. Springer, 2013.

[BSST13] Joshua Batson, Daniel A Spielman, Nikhil Srivastava, and Shang-Hua Teng. Spectral sparsification of graphs: theory and algorithms. *communications of the ACM*, 56(8):87–94, 2013.

[BY02] Yaneer Bar-Yam. General features of complex systems. *Encyclopedia of Life Support Systems (EOLSS), UNESCO, EOLSS Publishers, Oxford, UK*, 2002.

[BZ11] Vladimir Batagelj and Matjaž Zaveršnik. Fast algorithms for determining (generalized) core groups in social networks. *Advances in Data Analysis and Classification*, 5(2):129–145, 2011.

[CAI03] CAIDA. Caida skitter router-level topology measurements. 2003. http://www.caida.org/data/router-adjacencies/.

[CGH+13] Pilu Crescenzi, Roberto Grossi, Michel Habib, Leonardo Lanzi, and Andrea Marino. On computing the diameter of real-world undirected graphs. *Theoretical Computer Science*, 514:84–95, 2013.

[Cla05] A. Clauset. Finding local community structure in networks. *Physical Review E*, 72(2):26132, 2005.

[CN85] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.

[CN06] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5), 2006.

[CNM04] Aaron Clauset, Mark EJ Newman, and Cristopher Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[COJT+11] Luciano da Fontoura Costa, Osvaldo N Oliveira Jr, Gonzalo Travieso, Francisco Aparecido Rodrigues, Paulino Ribeiro Villas Boas, Lucas Antiqueira, Matheus Palhares Viana, and Luis Enrique Correa Rocha. Analyzing and modeling real-world phenomena with complex networks: a survey of applications. *Advances in Physics*, 60(3):329–412, 2011.

[CS11] Jie Chen and Ilya Safro. Algebraic distance on graphs. *SIAM Journal on Scientific Computing*, 33(6):3468–3490, 2011.

[CZF04] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-MAT: A recursive model for graph mining. *Computer Science Department*, page 541, 2004.

[CZG09] Jiyang Chen, Osmar Zaïane, and Randy Goebel. Local community identification in social networks. In *Proceedings of the 2009 International Conference on Advances in Social Network Analysis and Mining*, ASONAM '09, pages 237–242. IEEE Computer Society, 2009.

[Dat15] Dato. Website of the company distributing GraphLab, April 2015.

[DFC05] Jana Diesner, Terrill L Frantz, and Kathleen M Carley. Communication networks from the enron email corpus "it's always about the people. enron is no different". *Computational & Mathematical Organization Theory*, 11(3):201–228, 2005.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DKMZ11] Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Inference and phase transitions in the detection of modules in sparse networks. *Phys. Rev. Lett.*, 107:065701, Aug 2011.

[DRZ14] Naga Shailaja Dasari, Desh Ranjan, and Mohammad Zubair. ParK: An efficient algorithm for k-core decomposition on multicore processors. In Jimmy Lin, Jian Pei, Xiaohua Hu, Wo Chang, Raghunath Nambiar, Charu Aggarwal, Nick Cercone, Vasant Honavar, Jun Huan, Bamshad Mobasher, and Saumyadipta Pyne, editors, *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014*, pages 9–16. IEEE, 2014.

[dSPK79] Ithiel de Sola Pool and Manfred Kochen. Contacts and influence. *Social networks*, 1(1):5–51, 1979.

[EHR+08] Peter Ebbes, Zan Huang, Arvind Rangaswamy, Hari P Thadakamalla, and Oracle Retail Global Business Unit. Sampling large-scale social networks: Insights from simulated networks. In *18th Annual Workshop on Information Technologies and Systems, Paris, France*. Citeseer, 2008.

[EJRB13] David Ediger, Karl Jiang, E Jason Riedy, and David A Bader. GraphCT: Multithreaded algorithms for massive graph analysis. *Parallel and Distributed Systems, IEEE Transactions on*, 24(11):2220–2229, 2013.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, volume 96, pages 226–231, 1996.

[EMRB12] D. Ediger, R. McColl, J. Riedy, and D.A. Bader. STINGER: High performance data structure for streaming graphs. In *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*, pages 1–5, Sept 2012.

[Esd15] Kolja Esders. Link prediction in large-scale complex networks. Master's thesis, Karlsruhe Institute of Technology, http://parco.iti.kit.edu/attachments/Kolja%20Esders%20-%20Thesis.pdf, 2015.

[EW04] David Eppstein and Joseph Wang. Fast approximation of centrality. *J. Graph Algorithms Appl.*, 8:39–45, 2004.

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

[FB13] B. O. Fagginger Auer and R. H. Bisseling. Graph coarsening and clustering on the GPU. In David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors, *Graph Partitioning and Graph Clustering*, number 588 in Contemporary Mathematics. 2013.

[FBFM08] Santo Fortunato, Marián Boguñá, Alessandro Flammini, and Filippo Menczer. Approximating pagerank from in-degree. In *Algorithms and models for the web-graph*, pages 59–71. Springer, 2008.

[Fli14] Patrick Flick. Analysis of human tissue-specific protein-protein interaction networks. Master's thesis, Karlsruhe Institute of Technology, 2014.

[For] Santo Fortunato. Benchmark graphs to test community detection algorithms.

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

[Fre77] Linton C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, March 1977.

[Fre79] Linton C Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1979.

[Fre04] Linton Freeman. The development of social network analysis. *A Study in the Sociology of Science*, 2004.

[GD03] Pablo M. Gleiser and Leon Danon. Community structure in jazz. *Advances in Complex Systems*, 6(4):565–574, 2003.

[Gin12] Corrado Gini. Variabilità e mutabilità. *Reprinted in Memorie di metodologica statistica (Ed. Pizetti E, Salvemini, T). Rome: Libreria Eredi Virgilio Veschi*, 1, 1912.

[GKS+12] Robert Görke, Roland Kluge, Andrea Schumm, Christian Staudt, and Dorothea Wagner. An efficient generator for clustered dynamic random networks. *Design and Analysis of Algorithms*, pages 219–233, 2012.

[GLMY11] Ullas Gargi, Wenjun Lu, Vahab S Mirrokni, and Sangho Yoon. Large-scale community detection on youtube for topic discovery and exploration. In *International Conference on Weblogs and Social Media*, 2011.

[GN02] M. Girvan and M.E.J. Newman. Community structure in social and biological networks. *Proc. of the National Academy of Sciences*, 99(12):7821, 2002.

[Gör10] Robert Görke. *An Algorithmic Walk from Static to Dynamic Graph Clustering*. PhD thesis, Karlsruher Institut für Technologie, Dissertation, 2010.

[GPP12] Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. Random hyperbolic graphs: Degree sequence and clustering - (extended abstract). In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Proceedings, Part II*, pages 573–585, 2012.

[Gra73] Mark S Granovetter. The strength of weak ties. *American journal of sociology*, pages 1360–1380, 1973.

[GRS07] John R Gilbert, Steve Reinhardt, and Viral B Shah. High-performance graph algorithms from parallel sparse matrices. In *Applied Parallel Computing. State of the Art in Scientific Computing*, pages 260–269. Springer, 2007.

[GSM15] Alexander Gutfraind, Ilya Safro, and Lauren Ancel Meyers. Multiscale network generation. In *18th International Conference on Information Fusion, FUSION 2015, Washington, DC, USA, July 6-9, 2015*, pages 158–165, 2015.

[GSS08] Robert Geisberger, Peter Sanders, and Dominik Schultes. Better approximation of betweenness centrality. In *ALENEX*, pages 90–100. SIAM, 2008.

[GZFA10] Anna Goldenberg, Alice X Zheng, Stephen E Fienberg, and Edoardo M Airoldi. A survey of statistical network models. *Foundations and Trends® in Machine Learning*, 2(2):129–233, 2010.

[Hak62] S Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph. i. *Journal of the Society for Industrial & Applied Mathematics*, 10(3):496–506, 1962.

[Ham11] Zachary Hamaker. Electric networks and commute time. 2011.

[Hav55] Václav Havel. Poznámka o existenci konečných grafů. *Časopis pro pěstování matematiky*, 080(4):477–480, 1955.

[HLM+16] Michael Hamann, Gerd Lindner, Henning Meyerhenke, Christian L. Staudt, and Dorothea Wagner. Structure-preserving sparsification methods for social networks. *arxiv.org*, abs/1601.00286, 2016.

[HSSC08] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using NetworkX. Technical report, Los Alamos National Laboratory (LANL), 2008.

[IMSCR+08] Yasser Iturria-Medina, Roberto C Sotero, Erick J Canales-Rodríguez, Yasser Alemán-Gómez, and Lester Melie-García. Studying the human brain anatomical network via diffusion-weighted mri and graph theory. *Neuroimage*, 40(3):1064–1076, 2008.

[Jac09] Mathieu Jacomy. Force-atlas graph layout algorithm. *URL: http://gephi.org/2011/forceatlas2-the-new-version-of-our-home-brew-layout*, 2009.

[JCZB06] Pall F Jonsson, Tamara Cavanna, Daniel Zicha, and Paul A Bates. Cluster analysis of networks generated through homology: automatic identification of important protein communities involved in cancer metastasis. *BMC Bioinformatics*, 7:2, 2006.

[Joh02] David S Johnson. A theoretician's guide to the experimental analysis of algorithms. *Data structures, near neighbor searches, and methodology: fifth and sixth DIMACS implementation challenges*, 59:215–250, 2002.

[JS16] Emmanuel John and Ilya Safro. Single-and multi-level network sparsification by algebraic distance. *arXiv preprint arXiv:1601.05527*, 2016.

[JSYA+15] Mahdi Jalili, Ali Salehzadeh-Yazdi, Yazdan Asgari, Seyed Shahriar Arab, Marjan Yaghmaie, Ardeshir Ghavamzadeh, and Kamran Alimoghaddam. Centiserver: A comprehensive resource, web-based application and r package for centrality analysis. *PloS one*, 10(11):e0143111, 2015.

[Kap15] Andrea Kappes. *Engineering Graph Clustering Algorithms*. PhD thesis, Karlsruhe, Karlsruher Institut für Technologie (KIT), Diss., 2015, 2015.

[Kat53] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[KM15] Marcos Kiwi and Dieter Mitsche. A bound for the diameter of random hyperbolic graphs. *preprint available at http://arxiv. org/abs/1408.2947*, 2015.

[KN11] Brian Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Phys. Rev. E*, 83:016107, Jan 2011.

[KPCV14] Dániel Kondor, Márton Pósfai, István Csabai, and Gábor Vattay. Do the rich get richer? an empirical analysis of the bitcoin transaction network. *PloS one*, 9(2):e86197, 2014.

[KPK+10] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. Hyperbolic geometry of complex networks. *Physical Review E*, 82:036106, Sep 2010.

[KPPS13] Tamara G Kolda, Ali Pinar, Todd Plantenga, and C Seshadhri. A scalable generative graph model with community structure. *arXiv preprint arXiv:1302.6636*, 2013.

[KPS13] Kishore Kothapalli, SriramV. Pemmaraju, and Vivek Sardeshmukh. On the analysis of a label propagation algorithm for community detection. In Davide Frey, Michel Raynal, Saswati Sarkar, RudrapatnaK. Shyamasundar, and Prasun Sinha, editors, *Distributed Computing and Networking*, volume 7730 of *Lecture Notes in Computer Science*, pages 255–269. Springer Berlin Heidelberg, 2013.

[KR08] M.J. Keeling and P. Rohani. *Modeling infectious diseases in humans and animals*. Princeton University Press, 2008.

[KSV10] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for MapReduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 938–948. Society for Industrial and Applied Mathematics, 2010.

[KSVM15] Jannis Koch, Christian L. Staudt, Maximilian Vogel, and Henning Meyerhenke. Complex network analysis on distributed systems: An empirical comparison. In *International Symposium on Foundations and Applications of Big Data Analytics*, 2015.

[Kun13a] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 1343–1350. International World Wide Web Conferences Steering Committee, 2013.

[Kun13b] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 1343–1350. International World Wide Web Conferences Steering Committee, 2013.

[LAB+12] Adam Lugowski, David Alber, Aydın Buluç, John R. Gilbert, Steve Reinhardt, Yun Teng, and Andrew Waranis. A flexible open-source toolbox for scalable complex graph analysis. In *Proceedings of the Twelfth SIAM International Conference on Data Mining (SDM12)*, pages 930–941, April 2012.

[Lam10] Renaud Lambiotte. Multi-scale modularity in complex networks. In *Modeling and optimization in mobile, ad hoc and wireless networks (WiOpt), 2010 Proceedings of the 8th International Symposium on*, pages 546–553. IEEE, 2010.

[LBG+12] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

[LD10] J. Lin and C. Dyer. *Data-intensive Text Processing with MapReduce*. G - Reference,Information and Interdisciplinary Subjects Series. Morgan & Claypool, 2010.

[LF06] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 631–636, New York, NY, USA, 2006. ACM.

[LF07] Jure Leskovec and Christos Faloutsos. Scalable modeling of real graphs using kronecker multiplication. In *Proceedings of the 24th international conference on Machine learning*, pages 497–504. ACM, 2007.

[LF09a] Andrea Lancichinetti and Santo Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E*, 80(1):016118, 2009.

[LF09b] Andrea Lancichinetti and Santo Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Phys. Rev. E*, 80(1):016118, Jul 2009.

[LF11] Andrea Lancichinetti and Santo Fortunato. Limits of modularity maximization in community detection. *Physical review E*, 84(6):066122, 2011.

[LFR08] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4):046110, 2008.

[Lin14] Gerd Lindner. Complex network backbones. Master's thesis, Karlsruhe Institute of Technology, 2014.

[LK13] Dominique LaSalle and George Karypis. Multi-threaded graph partitioning. In *Proc. 27th IEEE Intl. Symposium on Parallel and Distributed Processing (IPDPS 2013)*, pages 225–236. IEEE Computer Society, 2013.

[LK14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[LK15] Dominique LaSalle and George Karypis. Multi-threaded modularity based graph clustering using the multilevel paradigm. *Journal of Parallel and Distributed Computing*, 76:66–80, 2015.

[LM12] Jure Leskovec and Julian J. Mcauley. 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, Inc., 2012.

[LS14] Jure Leskovec and Rok Sosič. SNAP: A general purpose network analysis and graph mining library in C++. http://snap.stanford.edu/snap, June 2014.

[LSB+03] David Lusseau, Karsten Schneider, Oliver J Boisseau, Patti Haase, Elisabeth Slooten, and Steve M Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

[LSH+15] Gerd Lindner, Christian L. Staudt, Michael Hamann, Henning Meyerhenke, and Dorothea Wagner. Structure-preserving sparsification of social networks. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2015, Paris, France, August 25 - 28, 2015*, pages 448–454, 2015.

[LWP08] F. Luo, J.Z. Wang, and E. Promislow. Exploring local community structures in large networks. *Web Intelligence and Agent Systems*, 6(4):387–400, 2008.

[MAB+10] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[McG12] Catherine C McGeoch. *A guide to experimental algorithmics*. Cambridge University Press, 2012.

[MCK12] Pádraig Mac Carron and Ralph Kenna. Universal properties of mythological networks. *EPL (Europhysics Letters)*, 99(2):28002, 2012.

[MIM15] Frank McSherry, Michael Isard, and Derek G Murray. Scalability! but at what cost? In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*, 2015.

[MKI+03] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. On the uniform generation of random graphs with prescribed degree sequences. *eprint arXiv:cond-mat/0312028*, December 2003.

[New02] Mark EJ Newman. Assortative mixing in networks. *Physical review letters*, 89(20):208701, 2002.

[New10] Mark Newman. *Networks: an introduction*. Oxford University Press, 2010.

[NLCB13] Bobo Nick, Conrad Lee, Pádraig Cunningham, and Ulrik Brandes. Simmelian backbones: Amplifying hidden homophily in facebook networks. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '13, pages 525–532, New York, NY, USA, 2013. ACM.

[NOB14] Arlind Nocaj, Mark Ortmann, and Ulrik Brandes. Untangling hairballs - from 3 to 14 degrees of separation. In Christian A. Duncan and Antonios Symvonis, editors, *Graph Drawing - 22nd International Symposium, GD 2014, Würzburg, Germany, September 24-26, 2014, Revised Selected Papers*, volume 8871 of *Lecture Notes in Computer Science*, pages 101–112. Springer, 2014.

[OB14] Mark Ortmann and Ulrik Brandes. Triangle listing algorithms: Back from the diversion. In Catherine C. McGeoch and Ulrich Meyer, editors, *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2014, Portland, Oregon, USA, January 5, 2014*, pages 1–8. SIAM, 2014.

[OFWB03] Joshua O'Madadhain, Danyel Fisher, Scott White, and Y Boey. The JUNG (java universal network/graph) framework. *University of California, Irvine, California*, 2003.

[OGS13] Michael Ovelgönne and Andreas Geyer-Schulz. An ensemble learning strategy for graph clustering. In David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors, *Graph Partitioning and Graph Clustering*, number 588 in Contemporary Mathematics. 2013.

[Ove13] Michael Ovelgönne. Distributed community detection in web-scale networks. In *Proc. Advances in Social Networks Analysis and Mining (ASONAM '13)*, pages 66–73, 2013.

[P. 60] A Rényi P. Erdős. On the Evolution of Random Graphs. *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, 1960.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[PDFV05] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.

[Pei06] Tiago P. Peixoto. graph-tool, 2006. http://graph-tool.skewed.de.

[PET+14] G Petri, P Expert, F Turkheimer, R Carhart-Harris, D Nutt, PJ Hellyer, and Francesco Vaccarino. Homological scaffolds of brain functional networks. *Journal of The Royal Society Interface*, 11(101):20140873, 2014.

[PGF02] Christopher R Palmer, Phillip B Gibbons, and Christos Faloutsos. Anf: A fast and scalable tool for data mining in massive graphs. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 81–90. ACM, 2002.

[PGO13] Fernando Perez, Brian E Granger, and CPSL Obispo. An open source framework for interactive, collaborative and reproducible scientific computing and education, 2013.

[PPPM+02] John J Potterat, L Phillips-Plummer, Stephen Q Muth, RB Rothenberg, DE Woodhouse, TS Maldonado-Long, HP Zimmerman, and JB Muth. Risk network structure in the early epidemic phase of hiv transmission in colorado springs. *Sexually transmitted infections*, 78(suppl 1):i159–i163, 2002.

[PSV+09] Symeon Papadopoulos, Andre Skusa, Athena Vakali, Yiannis Kompatsiaris, and Nadine Wagner. Bridge bounding: A local approach for efficient community discovery in complex networks. *arXiv preprint arXiv:0902.0871*, 2009.

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

[RAK07] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E*, 76(3):036106, 2007.

[RB13] E. Jason Riedy and David A. Bader. Multithreaded community monitoring for massive streaming graph data. In *IPDPS Workshops*, pages 1646–1655. IEEE, 2013.

[RBJ+11] Jason Riedy, David A. Bader, Karl Jiang, Pushkar Pande, and Richa Sharma. Detecting communities from given seeds in social networks. Technical Report GT-CSE-11-01, Georgia Institute of Technology, February 2011. Expanded from submitted version.

[RK15] Matteo Riondato and Evgenios M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery*, pages 1–38, 2015.

[RMEB13] E. Jason Riedy, Henning Meyerhenke, David Ediger, and David A. Bader. Parallel community detection for massive graphs. In David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, editors, *Graph Partitioning and Graph Clustering*, number 588 in Contemporary Mathematics. 2013.

[RN11] Randolf Rotta and Andreas Noack. Multilevel local search algorithms for modularity clustering. *J. Exp. Algorithmics*, 16:2.3:2.1–2.3:2.27, July 2011.

[San09] Peter Sanders. Algorithm engineering–an attempt at a definition. In *Efficient Algorithms*, pages 321–340. Springer, 2009.

[SB13] Julian Shun and Guy E. Blelloch. Ligra: a lightweight graph processing framework for shared memory. In Alex Nicolau, Xiaowei Shen, Saman P. Amarasinghe, and Richard W. Vuduc, editors, *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13, Shenzhen, China, February 23-27, 2013*, pages 135–146. ACM, 2013.

[SB15] Olaf Sporns and Richard F Betzel. Modular brain networks. *Annual review of psychology*, 67(1), 2015.

[SBV09] M. Ángeles Serrano, Marián Boguñá, and Alessandro Vespignani. Extracting the multiscale backbone of complex weighted networks. *Proceedings of the National Academy of Sciences*, 106(16):6483–6488, 2009.

[Sch07] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[SHH+10] Heli Sun, Jianbin Huang, Jiawei Han, Hongbo Deng, Peixiang Zhao, and Boqin Feng. gskeletonclu: Density-based network clustering via structure-connected tree division or agglomeration. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010.

[SHZ15] Wolfgang E. Schlauch, Emőke Ágnes Horvát, and Katharina A. Zweig. Different flavors of randomness: comparing random graph models with fixed degree sequences. *Social Network Analysis and Mining*, 5(1):1–14, 2015.

[SKL+10] Marcel Salathé, Maria Kazandjieva, Jung Woo Lee, Philip Levis, Marcus W Feldman, and James H Jones. A high-resolution human contact network for infectious disease transmission. *Proceedings of the National Academy of Sciences*, 107(51):22020–22025, 2010.

[SKP11] C. Seshadhri, Tamara G. Kolda, and Ali Pinar. Community structure and scale-free collections of Erdös-Renyi graphs. December 2011.

[SM13] Christian Staudt and Henning Meyerhenke. Engineering high-performance community detection heuristics for massive graphs. In *42nd International Conference on Parallel Processing, ICPP 2013, Lyon, France, October 1-4, 2013*, pages 180–189, 2013.

[SM16] Christian L. Staudt and Henning Meyerhenke. Engineering parallel algorithms for community detection in massive networks. *IEEE Transactions on Parallel and Distributed Systems*, 27(1):171–184, 2016.

[SN11] Jyothish Soman and Ankur Narang. Fast community detection algorithm with gpus and multicore architectures. In *Proc. 25th IEEE Intl. Parallel & Distributed Processing Symposium (IPDPS)*, pages 568–579. IEEE, 2011.

[SPR11] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 721–732, New York, NY, USA, 2011. ACM.

[SRD13] Tanwistha Saha, Huzefa Rangwala, and Carlotta Domeniconi. Sparsification and sampling of networks for collective classification. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 293–302. Springer, 2013.

[SRK16] Gupta Sukrit, Puzis Rami, and Kilimnik Konstantin. Comparative network analysis using kronfit. In *Complex Networks VII*, pages 363–375. Springer, 2016.

[ŠS06] Jiří Šíma and Satu Elisa Schaeffer. On the NP-completeness of some graph cluster measures. In *SOFSEM 2006: Theory and Practice of Computer Science*, pages 530–537. Springer, 2006.

[SSM+12] Christian Staudt, Andrea Schumm, Henning Meyerhenke, Robert Gorke, and Dorothea Wagner. Static and dynamic aspects of scientific collaboration networks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on*, pages 522–526. IEEE, 2012.

[SSP+14] Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M. Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey. Navigating the maze of graph analytics frameworks using massive graph datasets. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 979–990, New York, NY, USA, 2014. ACM.

[ST08] Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. *arXiv preprint arXiv:0809.3232*, 2008.

[ST15] Julian Shun and Kanat Tangwongsan. Multicore triangle computations without tuning. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2015.

[SW50] G. Simmel and K.H. Wolff. *The Sociology of Georg Simmel*. Free Press paperback. Free Press, 1950.

[SW05] Thomas Schank and Dorothea Wagner. Approximating clustering coefficient and transitivity. *Journal of Graph Algorithms and Applications*, 9(2):265–275, 2005.

[TBC+13] Yuanyuan Tian, Andrey Balmin, Severin Andreas Corsten, Shirish Tatikonda, and John McPherson. From "think like a vertex" to "think like a graph". *PVLDB*, 7(3):193–204, 2013.

[TGK] Sandia National Laboratories Tamara G. Kolda, Ali Pinar. Feastpack.

[TMP12] Amanda L Traud, Peter J Mucha, and Mason A Porter. Social structure of facebook networks. *Physica A: Statistical Mechanics and its Applications*, 391(16):4165–4180, 2012.

[UKBM11] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the facebook social graph. *arXiv preprint arXiv:1111.4503*, 2011.

[vLM15] Moritz von Looz and Henning Meyerhenke. Querying probabilistic neighborhoods in spatial data sets efficiently. *CoRR*, abs/1509.01990, 2015.

[vLMP15] Moritz von Looz, Henning Meyerhenke, and Roman Prutkin. Generating random hyperbolic graphs in subquadratic time. In *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, pages 467–478, 2015.

[VTX09] Konstantin Voevodski, Shang-Hua Teng, and Yu Xia. Finding local communities in protein networks. *BMC bioinformatics*, 10(1):297, 2009.

[WCWF03] Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. Epidemic spreading in real networks: An eigenvalue viewpoint. In *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on*, pages 25–34. IEEE, 2003.

[WHHF12] Ying-Jun Wu, Han Huang, Zhi-Feng Hao, and Chen Feng. Local community detection using link similarity. *Journal of Computer Science and Technology*, 27(6), 2012.

[XLJ+12] Bingying Xu, Zheng Liang, Yan Jia, Bin Zhou, and Yi Han. Local community detection using seeds expansion. In *Proceedings of the 2012 Second International Conference on Cloud and Green Computing*, CGC '12, pages 557–562, Washington, DC, USA, 2012. IEEE Computer Society.

[YL12] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, page 3. ACM, 2012.

[ZL09] R. Zafarani and H. Liu. Social computing data repository at ASU. 2009. http://socialcomputing.asu.edu.

APPENDICES

PUBLICATIONS

IN THIS THESIS

Some of the research leading to this thesis has appeared previously in the following publications.

Journal Articles

- Christian L. Staudt, Henning Meyerhenke: **Engineering Parallel Algorithms for Community Detection in Massive Networks** – *IEEE Transactions on Parallel and Distributed Systems*, January 2016
- Gerd Lindner, Christian L. Staudt, Michael Hamann, Henning Meyerhenke, Dorothea Wagner: **Structure-Preserving Sparsification Methods for Social Networks**. – to appear in *Social Network Analysis and Mining*, preprint available at http://arxiv.org/abs/1601.00286

Conference Papers

- Christian Staudt, Henning Meyerhenke: **Engineering High-Performance Community Detection Heuristics for Massive Graphs**. – *2013 International Conference on Parallel Processing (ICPP 2013)*, October 2013, Lyon, France
- Christian Staudt, Yassine Marrakchi, Henning Meyerhenke: **Detecting Communities around Seed Nodes in Complex Networks**. – *First International Workshop on High Performance Big Graph Data Management, Analysis, and Mining*, co-located with the *IEEE International Conference on Big Data (IEEE BigData 2014)*, October 2014, Washington DC, USA
- Gerd Lindner, Christian L. Staudt, Michael Hamann, Henning Meyerhenke, Dorothea Wagner: **Structure-Preserving Sparsification of Social Networks**. – *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2015)*, August 2015, Paris; France
- Jannis Koch, Christian L. Staudt, Maximilian Vogel, Henning Meyerhenke: **Complex Network Analysis on Distributed Systems: An Empirical Comparison**. – Best Paper Award at *International Symposium on Foundations and Applications of Big Data Analytics (FAB 2015)*, August 2015, Paris, France

Journal Articles in Revision Process

- Christian L. Staudt, Aleksejs Sazonovs, Henning Meyerhenke: **NetworKit: A Tool Suite for Large-scale Complex Network Analysis**. – for *Network Science*, preprint available at http://arxiv.org/pdf/1403.3005v3.pdf

OTHER PUBLICATIONS

The following publications appeared during the thesis period but are not contained in the thesis.

Journal Articles

- Robert Görke, Pascal Maillard, Andrea Schumm, Christian Staudt, Dorothea Wagner: **Dynamic graph clustering combining modularity and smoothness.** – *ACM Journal of Experimental Algorithmics*, Vol. 18, 2013

Conference Papers

- Elisabetta Bergamini, Henning Meyerhenke, Christian Staudt: **Approximating Betweenness Centrality in Large Evolving Networks**. – *SIAM Meeting on Algorithm Engineering and Experimentation (ALENEX 2015)*, January 2015, San Diego, USA

Informal

- Moritz von Looz, Christian L. Staudt, Henning Meyerhenke, Roman Prutkin: **Fast generation of dynamic complex networks with underlying hyperbolic geometry**. – *arxiv.org abs/1501.03545*, 2015
- Roland Glantz, Christian L. Staudt, Henning Meyerhenke: **Correspondences between partitions**. – *arxiv.org/abs/1603.04788*, 2016, under review for *Mathematical Programming*

Christian L. Staudt

Curriculum Vitae

born October 21st 1985 in Mainz

Education

2012 – 2016 **Karlsruhe Institute of Technology (KIT)**, *PhD student at the Institute of Theoretical Informatics/Parallel Computing Group.*
– advisor: Juniorprof. Dr. Henning Meyerhenke

2005 – 2012 **Karlsruhe Institute of Technology (KIT)**, *computer science studies.*
– **Diplom** 2012

2000 – 2004 **Kolleg St. Blasien**, *St. Blasien.*
Abitur 2004

1996 – 2000 **Kurfürst-Balduin-Gymnasium**, *Münstermaifeld.*

Related Professional Experience

2012 – 2016 **Researcher**, *(E13), at the Institute of Theoretical Informatics/Parallel Computing Group*, KIT.

2007 – 2010 **Student research assistant**, *at the Institute of Theoretical Informatics, research group Algorithmics I (Prof. Dorothea Wagner)*, KIT.
– working with Robert Görke and Andrea Kappes

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of July 7, 2016 (`classicthesis` version 0.2).

ERKLÄRUNG

Ich versichere, diese Dissertation selbstständig angefertigt, alle benutzten Hilfsmittel vollständig angegeben, die Satzung des Karlsruher Instituts für Technologie (KIT) zur Sicherung guter wissenschaftlicher Praxis beachtet, und kenntlich gemacht zu haben, was aus Arbeiten anderer und eigener Veröffentlichungen unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 2016

Christian Lorenz Staudt